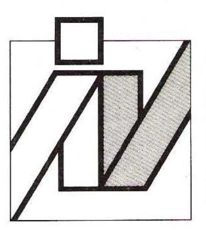
Hans Lorenz Schneider

Neue Möglichkeiten mit dem Commodore 64/128

durch anwenderfreundliche Musterprogramme, Anleitungen zur erfolgreichen Programmierung und Erweiterungen

- Programmierkurse für Basic, Pascal, Forth, Logo, Assembler
- Neue Musterprogramme und Hilfsroutinen für Wirtschaft, Technik, Graphik und Sound
- Detaillierte Systembeschreibungen
- Bauanleitungen mit Platinenfolien und Programme für Erweiterungen u. Zubehör
- Programmierhilfen und Praxistips





INTEREST-VERLAG
Fachverlag für
Special Interest Publikationen
und Anwendersoftware
Industriestraße 21
D-8901 Kissing
Tel. 08233/2101-0

Sehr geehrter Leser!

Herzlich willkommen im Kreis der Leser unseres Commodore 64/128-Nachschlagewerkes, das nun in 2., erweiterter und aktualisierter Auflage mit zwei Grundwerksdisketten erschienen ist.

Ihr Vorteil: Sie verfügen jetzt über einen vielseitigen Ratgeber, wenn es um Fragen des Einsatzes Ihres Commodores geht.

Das Werk bietet Ihnen:

- hundertprozentig lauffähige Programme, z.B. das menügesteuerte Tabellenkalkulationsprogramm "Aliplan"
- unentbehrliche Utilities, Tricks und Tips für erfolgreiches Programmieren
- Kurse und Anwendungen für Grafik und Sound
- rechnerbezogene Programmierkurse für höhere Programmiersprachen und Assembler
- genaue Beschreibung von Hard- und Software Ihres Commodores
- komplette Bauanleitungen, u.a. für einen parallelen IEC- Anschluß
- Know-how für spezielle Einsatzbereiche, z.B. zur Modellbahnsteuerung

Nehmen Sie das Werk in die Hand, blättern Sie es durch und überzeugen Sie sich selbst.

Dieses Werk veraltet nicht

Aus eigener Erfahrung wissen Sie selbst am besten, welche Vielzahl von Möglichkeiten Ihnen Ihr Commodore bietet. Hinzu kommen die Neuerungen, die laufend die "Commodore-Welt" erweitern, Programmiersprachen, zusätzliche Peripherie usw.

Wir haben uns daher entschlossen, speziell für dieses Werk einen Erweiterungsservice anzubieten. Er versorgt Sie alle 2 - 3 Monate zuverlässig mit detaillierten Hardwarebeschreibungen, neuen Programmen und Routinen, praktischen Programmier- und Anwendungskursen.

Praxishandbuch + Disketten

Zusammen mit dem Nachschlagewerk erhalten Sie einen kompletten 6510-Assembler, das menügesteuerte Tabellenkalkulationsprogramm "Aliplan", die Mausprogrammierung und das Spiel "Alpha V". Assembler, Disassembler und Monitor auf Diskette ermöglichen Ihnen sofort das Programmieren in Assembler.

Unser Diskettenservice bietet Ihnen darüber hinaus die Möglichkeit, auch die Programme der Erweiterungen zu beziehen.

Vorschau auf die nächsten Erweiterungen

Es sind Beiträge zu folgenden Themenbereichen vorgesehen:

- Hardwarebeschreibungen und weitere Bauanleitungen z.B. zur RS 232-Schnittstelle oder dem Thema "Sprachausgabe" mit dem Commodore
- Tips, Tricks und Utilities für Einsteiger und Profis z.B. zur Beschleunigung der Floppy, Erweiterungen des Betriebssystem, Manipulation von Farben und Zeichensätzen
- Grafik und Sound: Kurse und Beispiele für 3D-Effekte und Raytracing
- Musterlösungen für den Bereich Wirtschaft und Verwaltung z.B. Lagerverwaltung und Kassenbuchführung
- Spezielle Einsatzbereiche, wie z.B. DFÜ (alles über Mailboxen) oder Drucken mit BTX

Teilen Sie uns bitte Ihre Wünsche mit:

Wir wollen dieses Werk auch in Zukunft genau auf Ihre Bedürfnisse zuschneiden. Deshalb an dieser Stelle die Aufforderung an Sie, unseren engagierten Leser: Bitte schreiben Sie uns, welche Themen Sie besonders interessieren und zu welchen Bereichen Sie gern Programme haben möchten.
Unsere Aufgabe wird es sein, Ihre Wünsche so schnell wie möglich zu erfüllen.

Wir bedanken uns schon vorab für die Zusammenarbeit.

Viel Erfolg mit Ihrem neuen Handbuch wünscht Ihnen

Monika Köhler

Monika Köhler
Produktmanagerin für Informatik

Neue Möglichkeiten mit dem Commodore 64 (128)

**Programmierkurse für Basic, Pascal, Forth, Logo, Assembler;
Neue Musterprogramme und Hilfsroutinen für Wirtschaft, Technik, Graphik und Sound;
Programmierhilfen und Praxistips;
Detaillierte Systembeschreibungen;
Bauanleitungen und Platinenfolien und Programme für Erweiterungen und Zubehör**

Herausgegeben von
Hans Lorenz Schneider

Interest-Verlag GmbH + Co. KG · Fachverlag für Special Interest Publikationen und Anwendersoftware

Dieses Werk wurde mit der größtmöglichen Sorgfalt zusammengestellt. Dennoch kann nicht völlig ausgeschlossen werden, daß sich im Text, in den Listings, den Schaltplänen, den Platinen-Layouts oder den gelieferten Programmen Fehler bzw. Unklarheiten eingeschlichen haben.

Der Verlag leistet deshalb im Rahmen der zivilrechtlichen Vorschriften Gewähr für die Vollständigkeit und einwandfreie Beschaffenheit des gelieferten Werkes. Für einzelne inhaltliche Fehler übernimmt der Verlag keine Gewährleistung. Darüber hinaus haften Verlag, Herausgeber und Autoren für Folgeschäden nur dann, wenn ihnen Vorsatz oder grobe Fahrlässigkeit nachgewiesen werden kann.

Die Realisierung der Projekte aus unseren Werken erfolgt auf eigene Gefahr.

Für Hinweise auf Fehler und für die Zusendung von Verbesserungsvorschlägen sind Verlag und Herausgeber dankbar. Der Verlag wird in solchen Fällen bemüht sein, Fehler oder Unklarheiten auszuräumen.

Beim Nachbau von elektronischen Schaltungen werden Grundkenntnisse über die Behandlung der Bauteile und den Umgang mit elektronischen bzw. elektrischen Geräten vorausgesetzt. Vor der Inbetriebnahme ist sicherzustellen, daß eine Berührung netzspannungsführender Teile ausgeschlossen ist. Vor Eingriffen in elektrische Geräte sind diese unbedingt durch Ziehen des Netzsteckers vom Netz zu trennen. Arbeiten an Teilen, an denen während des Betriebs Netzspannung anliegt, dürfen nur vom Fachmann vorgenommen werden. In Zweifelsfällen ist immer fachmännischer Rat einzuholen. Es ist zu beachten, daß die Bildröhren von Fernsehgeräten und Monitoren auch nach Ziehen des Netzsteckers noch unter gefährlicher Hochspannung stehen können. Beim Umgang mit technischen Geräten und Chemikalien sind in jedem Falle auch die Hinweise des Herstellers zu beachten. Bei der Realisierung von Projekten aus unseren Werken sind unbedingt auch Korrekturen und Ergänzungen des betreffenden Beitrages in späteren Ergänzungslieferungen zu beachten.

Die in diesem Werk veröffentlichten Texte, Schaltpläne, Verfahren, Programme und Warnungen werden ohne Rücksicht auf einen eventuellen Patentschutz oder andere Rechte wiedergegeben.

Herausgeber: Hans Lorenz Schneider
Printed in Germany 1989
Bestell-Nr.: 2000

Teil 1

Inhalt

1

Inhalt

Kapitelnachweis und Detailübersicht zum Grundwerk Neue Möglichkeiten mit dem Commodore 64/128

Teil 2

Hardwarebeschreibung

2/1 Äußerer Aufbau

Im Rahmen der Hardwarebeschreibung wollen wir als erstes das Thema behandeln, was jedem quasi sofort ins Auge sticht, der äußere Aufbau. Neben der Tastatur werden in diesem Kapitel die einzelnen Steckanschlüsse sowohl für den C 64 als auch den C 128 PC beschrieben.

2/1.1 Äußerer Aufbau des C 64

Da das Werk sich größtenteils mit dem C 64 befaßt, wollen wir ihn auch zuerst im Rahmen des äußeren Aufbaues besprechen und mit den Anschlüssen beginnen.

2/1.1.1 Anschlüsse

Die vielen Anschlußmöglichkeiten von Peripherie- und Zusatzgeräten machen einen entscheidenden Vorteil des C 64 aus. Neben dem obligatorischen Netzanschluß sind

zunächst zwei wichtige Anschlüsse zu nennen: Die Control-Ports und der IEC-Bus. Die *Control-Ports* erlauben den Anschluß von Joysticks, Paddels oder auch eines Light-Pen sowie neuerdings auch „Mäuse". Da der C 64 hauptsächlich als Spielgerät benutzt wird, kommt dieser in Kapitel 2/1.1.1.2 beschriebene Anschlußmöglichkeit eine sehr wichtige Funktion zu.

Der *IEC-Bus* wird benötigt um die Peripheriegeräte wie z.B. Floppy und Drucker, anzuschließen. Dabei weisen die Peripheriegeräte je zwei solcher gleichartigen Anschlüsse auf, so daß die Verbindung von Computer zum ersten Peripheriegerät und von dort zum zweiten und von dort zu einem weiteren Peripheriegerät gezogen werden kann. Mit dem IEC-Bus werden wir uns in Kapitel 2/1.1.1.5 beschäftigen.

Ein weiterer wichtiger Anschluß ist der *Expansion-Port,* der hauptsächlich für Standard-Software verwendet wird, um sich das Laden von Diskette zu ersparen. Am Expansion-Port angeschlossene Module betrachtet der Rechner in gewisser Weise als integriert und führt in der Regel gleich nach dem Einschalten des Rechners ein dort befindliches Programm aus. Daß der Expansion-Port jedoch nicht nur für käufliche Module nutzbar ist, wollen wir im Rahmen dieses Buches aufzeigen, indem wir z.B. unseren Assembler auf ein solches Modul auslagern und auch eine Betriebssystem-Erweiterung soll dort plaziert werden.

Für Anwender, die mit ihrem C 64 Meßvorgänge erfassen wollen, aber auch für Bastler mit eigenen Hardware-Ergänzungen ist der *User-Port* von besonderer Bedeutung. Neben den in Kapitel 2/1.1.1.3 vorgestellten allgemeinen Merkmalen dieser Anschlußmöglichkeit werden wir ihm im Rahmen dieses Buches noch an einigen Stellen begegnen.

Der *Cassetten-Port* ist nur für diejenigen Betreiber eines C 64 interessant, die (noch) mit einem Kassettenrekorder arbeiten. Die Aufstellung der Anschlüsse wird vervollständigt durch die Beschreibung des *Antennenanschlusses* und die Verbindungen zum *Monitor.*

2/1.1.1.2

Control-Ports

Die beiden Control-Ports 1 und 2 dienen hauptsächlich zum Anschluß von Joysticks, Paddles oder eines Light-Pen. Aber auch andere Anwendungen sind denkbar (z.B. Maus), wobei der User-Port jedoch wesentlich einfacher zu handhaben ist.

Der neunpolige Anschluß entspricht der Quasi-Norm, die die Firma Atari zuerst für ihre Joysticks benutzte. Die Kontakte sind in zwei Reihen angeordnet, wobei der Stecker die Form eines abgerundeten Trapezes aufweist:

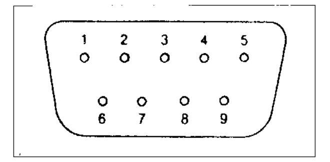


Bild 2/1.1.1.2
Steckanschluß Control-Ports 1 und 2

Bei den Paddles (zwei Paddles an einem Anschlußstecker) ist zu beachten, daß jeweils nur ein Paddlepaar angesteuert werden kann, da die Information analog verarbeitet werden muß (Drehwiderstand) und dies vom C 64 nur über die Pins 23 und 24 des SID zu realisieren ist. Näheres dazu siehe in Kapitel 2/3.6.

Außer den beiden Anschlüssen 5 und 9, die die eben beschriebene analoge Information an die Pins 23 und 24 des SID übergeben und der Stromversorgung an den Anschlüssen 7 und 8 sind alle Pins beider Control-Ports mit dem CIA verbunden. Näheres geht aus folgender Tabelle hervor.

Control-Port 1		
Pin	Funktion	verbunden mit
1	Joystick 1 nach oben	CIA PB0
2	Joystick 1 nach unten	CIA PB1
3	Joystick 1 nach links	CIA PB2
4	Joystick 1 nach rechts	CIA PB3
5	Paddlepaar 1 Paddle 1	SID Pin 23 über Analogschalter
6	Feuerknopf Joystick 1 oder Light-Pen	CIA PB4 und VIC LP*
7	+5 V (bis 100 mA)	(Stromversorgung)
8	Masse	(Stromversorgung)
9	Paddlepaar 1 Paddle 2	SID Pin 24 über Analogschalter

Control-Port 2		
Pin	Funktion	verbunden mit
1	Joystick 2 nach oben	CIA PA0
2	Joystick 2 nach unten	CIA PA1
3	Joystick 2 nach links	CIA PA2
4	Joystick 2 nach rechts	CIA PA3
5	Paddlepaar 2 Paddle 1	SID Pin 23 über Analogschalter
6	Feuerknopf Joystick 2	CIA PA und VIC LP
7	+5 V (bis 100 mA)	(Stromversorgung)
8	Masse	(Stromversorgung)
9	Paddlepaar 2 Paddle 2	SID Pin 24 über Analogschalter

Die Pins 1 bis 4 und 6 werden standardmäßig als digitaler Eingang verwendet, können jedoch auch als digitaler Ausgang angesprochen werden. Das Prinzip der Joysticks ist recht einfach, da je nach Bewegung ein bestimmter Kontakt im Joystick mit Masse kurzgeschlossen wird. Joysticks, die auch Diagonale zulassen, schließen dementsprechend gleichzeitig zwei Kontakte mit Masse kurz.

Die Paddles bringen einen Widerstand zwischen dem ihnen zugeordneten Anschluß (Pin 5 oder 9) und die Stromversorgung (Pin 7). Um optimale Ergebnisse zu erzielen, sollte ein Drehwiderstand im Bereich 200 Ohm bis 200 KOhm eingesetzt werden. Siehe auch Kapitel 2/3.5 (Joystick) und 7/2.6 (Selbstbau eines Light-Pens).

2/1.1.1.3

User-Port

Der User-Port ist die wohl am häufigsten verwendete Schnittstelle, wenn man externe Geräte steuern oder Meßwerte erfassen will. Innerhalb dieses Buches werden wir noch häufiger auf den User-Port zurückgreifen, wobei er jedoch nicht immer direkt programmiert werden muß. Z.B. der Anschluß des Märklin Digital H0 Interface (siehe Teil 8) wird am User-Port vorgenommen.

Der User-Port ist zugleich die hardwaremäßig am leichtesten zu handhabende Schnittstelle. Auch die softwaremäßig realisierte RS 232-Schnittstelle (auch als V 24 bekannt) wird über den User-Port geführt.

Die leichte Handhabbarkeit des User-Ports ergibt sich daraus, daß die meisten Leitungen mit dem Complex Interface Adapter (CIA) verbunden sind (siehe Kapitel 2/2.1.5). Aus der weiter unten aufgeführten Tabelle geht die Verbindung der einzelnen Anschlüsse hervor. Zunächst wollen wir uns jedoch den Anschluß selbst betrachten.

Bild 2/1.1.1.3-1 zeigt die Anschlüsse des User-Ports mit ihren Bezeichnungen.

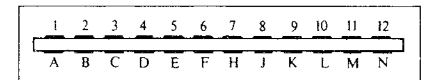


Bild 2/1.1.1.3-1
Steckanschluß des User-Ports

Der User-Port ist als 24-poliger Stecker ausgebildet, wovon die vier äußeren Anschlüsse (1, 12, A, N) alle Masse (GND) führen. Drei weitere Anschlüsse dienen zur Stromversorgung externer Geräte (2, 10, 11). Die restlichen 17 Anschlüsse weisen eine digitale Funktion auf, die aus folgender Tabelle hervorgeht:

Pin	Bezeichnung	Funktion
1	GND	Masse
2	+5V	Stromversorgung für externe Geräte (100 mA)
3	RESET	entsprechende Leitung des Prozessors
4	CNT1	CNT-Anschluß des CIA 1
5	SP1	SP-Anschluß des CIA 1
6	CNT2	CNT-Anschluß des CIA 2
7	SP2	SP-Anschluß des CIA 2
8	PC2	PC-Anschluß des CIA 2 (geht für einen Takt auf Low, wenn ein Schreib- oder Lesezugriff auf das Portregister B ausgeführt wurde)
9	ATN	ATN des seriellen Bus

Pin	Bezeichnung	Funktion
10	9 V AC	Wechselstrom für externe Geräte
11	9 V AC	Wechselstrom für externe Geräte
12	GND	Masse
A	GND	Masse
B	FLAG2	FLAG-Anschluß des CIA 2
C	PB0	Port B Bit 0 von CIA 2
D	PB1	Port B Bit 1 von CIA 2
E	PB2	Port B Bit 2 von CIA 2
F	PB3	Port B Bit 3 von CIA 2
H	PB4	Port B Bit 4 von CIA 2
J	PB5	Port B Bit 5 von CIA 2
K	PB6	Port B Bit 6 von CIA 2
L	PB7	Port B Bit 7 von CIA 2
M	PA2	Port A Bit 2 von CIA 2
N	GDN	Masse

Bis auf die Stromversorgungsanschlüsse und die Anschlüsse 3 und 9 sind alle Pins mit den beiden CIA's verbunden, in der Hauptsache mit CIA 2.

Der Rechner selbst greift auf einige Leitungen und Funktionen der CIA's zurück, was bei einer Programmierung zu berücksichtigen ist. Belegt sind u.a. die Ports A und B sowie die Leitungen FLAG und Timer A beim CIA 1 und PA0 und PA1, sowie PA3 bis PA7 bei CIA 2.

Bei Verwendung der RS 232-Schnittstelle wird der CIA 2 fast ausschließlich durch diese belegt, so daß hier eine Eigenprogrammierung nicht mehr möglich ist.

Auf Beispiele wollen wir an dieser Stelle verzichten, da wir den User-Port mehrfach innerhalb des Buches anwenden werden.

2/2
Interner Aufbau

Nachdem wir den C 64 und den C 128 von außen unter die Lupe genommen haben, wollen wir uns mit dem Innenleben dieser beiden Rechner beschäftigen. Auch im Kapitel 2/2 werden wir wieder beide Rechner getrennt behandeln.

2/2.1
Interner Aufbau des C 64

Von keinem Rechner ist das Innenleben so gut bekannt wie vom C 64. Neben einer allgemeinen Einführung, die auch ein Blockschaltbild beinhaltet, um die Zusammenhänge aufzuzeigen, wollen wir die einzelnen operativen Einheiten des C 64 besprechen. Die CPU 6510 ist eine Nachfolge-CPU der CPU des legendären PET, der eine 6502 CPU hatte. Sie wurden übrigens von der Commodore-Tochterfirma MOS Technology entwickelt.

Für die Bearbeitung des Bildschirmes besitzt der C 64 einen eigenen Video-Controller, VIC (Video Interface Chip) genannt, der genauso wie der Sound-Chip (SID) aus der 65xx-Familie stammt.

Für die Verbindung nach außen sorgen im C 64 zwei sogenannte Complex Interface Adapter, auch kurz CIA genannt. Abschließend gehen wir auf das logische Programmierfeld ein (PLA), das hauptsächlich der Steuerung der Bausteine untereinander dient.

2/2.1.1

Einführung mit Blockschaltbild

Das Bild zeigt die Architektur des C 64. Die Zentraleinheit des Computers bildet die CPU 6510, ein von Commodore entwickelter Baustein, der eine Variante der weitverbreiteten CPU (Zentraleinheit) 6502 ist. Die CPU kontrolliert alle Einheiten des Systems. Dies geschieht über drei Arten von Leitungen:

— den Adreßbus, über den die gewünschte Speicherstelle oder der gewünschte Baustein ausgewählt wird. Im C 64 bilden 16 Leitungen den Adreßbus.

— den Datenbus, über den die Daten aus Speicherstellen gelesen oder in Speicherstellen geschrieben werden. Im C 64 bilden 8 Leitungen den Datenbus, er gehört daher zu den 8-Bit-Computern.

— die Steuerleitungen. Zu diesen gehört eine Reihe von Leitungen, die z.B. die Datenrichtung bestimmen (Lesen oder Schreiben von Daten), anzeigen, ob eine gültige Adresse auf dem Adreßbus anliegt und ähnliches.

Leider kann die CPU mit ihren 16 Adreßleitungen nur 64 KByte (= 65536 verschiedene Adressen) Speicher adressieren. Der C 64 verfügt aber über 64 KByte Schreib-Lesespeicher (RAM), 20 KByte Nur-Lesespeicher (ROM) sowie über einen 4 KByte großen Speicherbereich (I/O), in dem die Bausteine für die Ein- und Ausgabe auf Bildschirm, Tastatur etc. liegen. Um dies alles ansprechen zu können, entscheidet die Adreßdekodierung, welche Einheiten in welchen Adreßbereichen angesprochen werden. Die hierbei bestehenden Möglichkeiten entnehmen Sie bitte dem Kapitel 3/1.1 „Übersicht und Speicherbelegung".

Alle im Speicherbereich des Computers liegenden Bausteine verfügen über einen CS-Eingang (CS = Chip Select, engl. Bausteinauswahl). Über diesen Eingang wird dem Baustein mitgeteilt, ob er angesprochen ist. Ist dies nicht der Fall, verhält sich der Baustein, als wäre er nicht vorhanden. Normalerweise geht ein Zugriff der CPU auf das RAM des C 64, welches mit seinen 64 KByte den gesamten Adreßraum belegt. Entscheidet die Adreßdekodierung jedoch anders, blendet sie den RAM-Bereich aus, und wählt einen der ROM-Bausteine oder den I/O-Bereich über das CS-Signal aus.

Der ROM-Bereich selbst ist in drei Teile geteilt. In den ersten beiden Bausteinen liegen der Basic-Interpreter sowie das Betriebssystem des Computers, wobei sich der Basic-Interpreter noch im Betriebssystem-ROM fortsetzt. Diese beiden Bausteine werden normalerweise von der Adreßdekodierung in den Speicherbereich eingeblendet. Die restlichen 4 KByte ROM beinhalten den Zeichensatz des C 64 (genau ge-

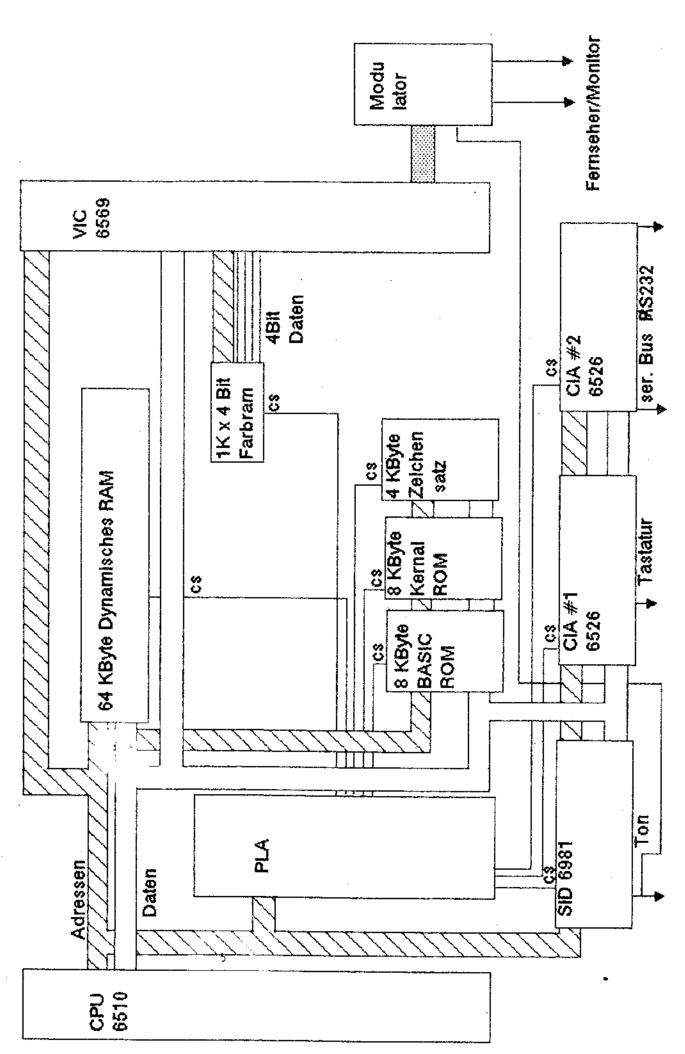


Bild 2/2.1-1 Blockschaltbild des C 64

nommen sind es zwei Zeichensätze, einer für Großschrift/Graphik und einer für Klein- und Großschrift). Er wird von der Adreßdekodierung besonders behandelt. Wir werden darauf noch bei der Beschreibung des Video-Bausteins eingehen.

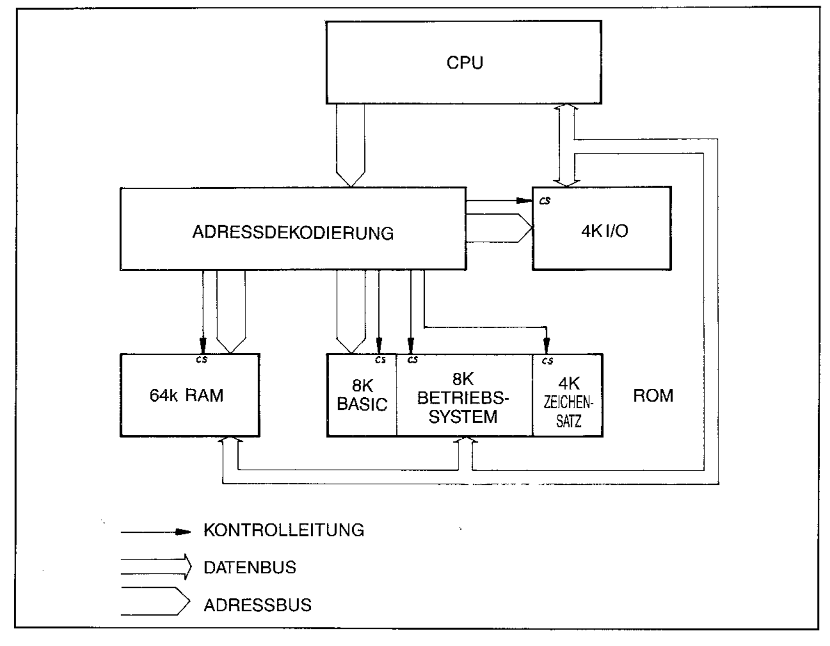


Bild 2/2.1.-2 Die Architektur des C 64

In dem 4 KByte großen I/O-Bereich liegen die Ein- und Ausgabebausteine. Beim C 64 sind dies der Video-IC (VIC), der Sound-IC (SID) sowie zwei universelle Ein- und Ausgabebausteine (CIA). Die Bausteine enthalten jeweils eine unterschiedliche Anzahl von Registern, über die ihre Funktionen gesteuert werden. Die CPU kann direkt auf diese Register, wie auf normale Speicherzellen, zugreifen. Ausgewählt werden die einzelnen Bausteine dann von der Adreßdekodierung.

Die Signale des Sound-Bausteins SID und des Video-Bausteins VIC werden gemeinsam einem Modulator zugeführt. Dieser arbeitet wie ein kleiner Sender: Er macht aus den Eingangssignalen ein Sendesignal, welches von normalen Fernsehern verarbeitet werden kann.

Einen besonderen Status bei den Ein- und Ausgabegeräten nimmt der Kassettenrekorder ein. Er wird nicht über einen der Ein- und Ausgabe-Bausteine betrieben,

sondern direkt von der CPU. Die CPU 6510 verfügt über sechs unabhängige Ein- und Ausgabeleitungen, von denen drei Leitungen die Steuerung der Datasette übernehmen.

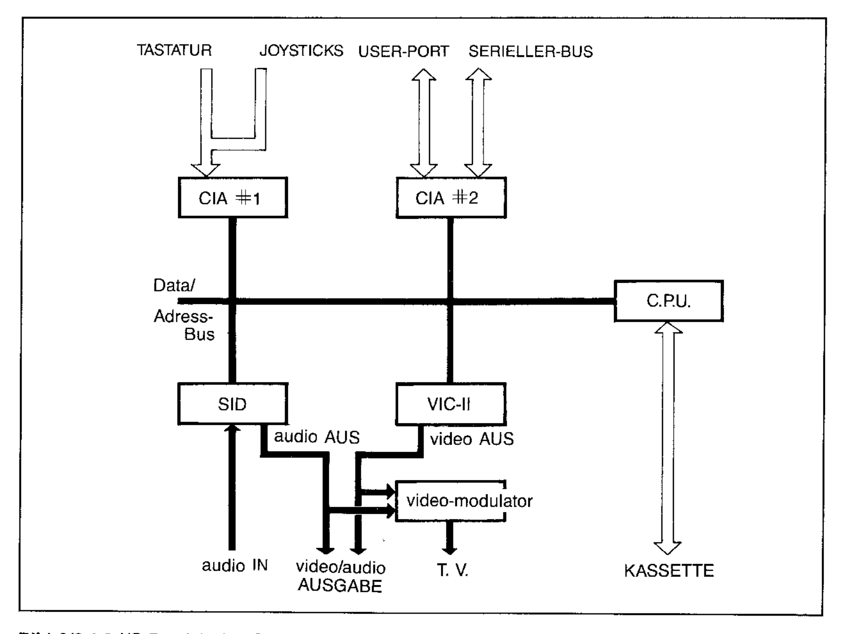


Bild 2/2.1-3 I/O-Bereich des C 64

In den folgenden Kapiteln werden nun die Bausteine des C 64 näher betrachtet. Die zu den Bausteinen VIC, SID und CIA gehörenden Register finden Sie im Kapitel 3/1.2 beschrieben.

2/2.1.2

Der Prozessor 6510

Der Prozessor 6510 ist die Zentraleinheit des C 64. Er steuert alle Abläufe im Computer. Zu diesem Zweck verfügt er über 16 Adreßleitungen zur Auswahl der Speicherstelle, acht Datenleitungen, über die Werte aus Speicherstellen gelesen oder in sie geschrieben werden können, sowie sechs frei programmierbare Leitungen und einige Steuerleitungen. Die folgende Tabelle zeigt die Pinbelegung und ihre Bedeutung.

Pin	Name	Beschreibung
1	PHO In	Eingang des Systemtakts ca. 980 KHz
2	RDY	Ready-Leitung
3	/IRQ	Interrupt-Anforderung (**I**nterrupt **Req**uest)
4	/NMI	Interrupt-Anforderung (**N**on **m**askable **I**nterrupt)
5	/DMA	DMA-Eingang
6	VCC	Versorgungsspannung: +5V
7-20, 22, 23	AO..A15	Adreßleitungen
21	GND	Versorgungsspannung: Masse
24-29	P5..P0	I/O Leitungen
30-37	D7..DO	Datenleitungen
38	R/W	Lese-Schreibleitung
39	PH2 Out	Systemtakt-Ausgang wird 1 bei gültiger Adresse
40	/RESET	Rücksetzen des Prozessors

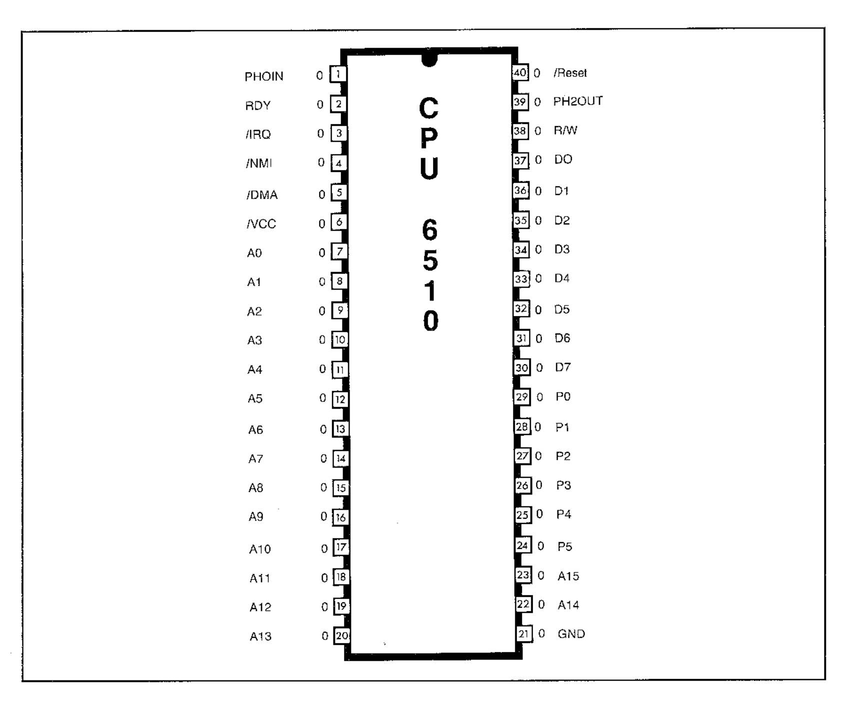
Einige der Leitungen sind es Wert, etwas genauer betrachtet zu werden:

/IRQ und /NMI (Pin 3 und 4)

Dies sind Leitungen, die den Prozessor zur Unterbrechung seiner Arbeit bewegen. Der genaue Ablauf kann im Kapitel 4/5.2.1.3 nachgelesen werden. Für die Erzeugung der Signale ist je einer der beiden CIAs zuständig. Außerdem können SID und VIC ein IRQ-Signal erzeugen.

/DMA (Pin 5)

Über diesen Eingang ist es möglich, die CPU 6510 vollständig abzuschalten. Da der Eingang vom Modulport kommt, kann so eine externe CPU (z.B. die CPU Z-80 der CP/M-Karte) die Kontrolle über die Komponenten des Computers übernehmen.



P0 bis P5 (Pin 24-29)

Diese Leitungen bilden den Prozessor-Port. Im C 64 haben sie folgende Bedeutung:

Leitung	Datenrichtung	Funktion
0	Ausgabe	Signal LORAM an PLA
1	Ausgabe	Signal HIRAM an PLA
2	Ausgabe	Signal CHAREN an PLA
3	Ausgabe	Schreibleitung an Datasette
4	Eingabe	Abfrageleitung der Recordertaste
5	Ausgabe	Ausschalten des Recordermotors

Die Programmierung der Leitungen ist im Kapitel 4/5.2.1.4 nachzulesen.

R/W (Pin 38)

Entscheidet, ob es sich um einen Lese- oder Schreibzugriff handelt. Bei einem Schreibzugriff legt der Prozessor die Leitung auf logisch 0.

2/2.1.3

Der Sound-Generator 6581 (SID)

Der Sound-IC SID übernimmt die Erzeugung der Geräusche oder von Musik. Er verfügt über drei unabhängige Stimmen, die miteinander synchronisiert werden können. Für jede Stimme kann eine Hüllkurve sowie die Wellenform bestimmt werden. Zusätzlich stehen noch Filter sowie ein Ton-Eingang zur Verfügung, über den von außen Tonsignale zugeführt werden können. Für den SID gilt die folgende Pinbelegung:

Pin	Name	Beschreibung
1, 2	CAP1	Anschluß eines Kondensators für den Filter. Im C 64 beträgt die Kapazität 2200 pF.
3, 4	CAP2	Siehe Pin 1, 2.
5	/RESET	Reset-Leitung.
6	PH2	Taktleitung vom Prozessor.
7	R/W	R/W-Leitung vom Prozessor.
8	/CS	Baustein Auswahlsignal von der Adreßdekodierung.
9-13	A0..A4	Adreßleitungen zur Auswahl eines Registers.
14	GND	Versorgungsspannung: Masse.
15-22	D0..D7	Datenleitungen.
23	POTX	Analog-Eingabe.
24	POTY	Analog-Eingabe.
25	VCC	Versorgungsspannung: +5V.
26	EXTIN	Externer Toneingang.
27	OUT	Tonausgang zum Modulator.
28	VDD	Versorgungsspannung: +12V.

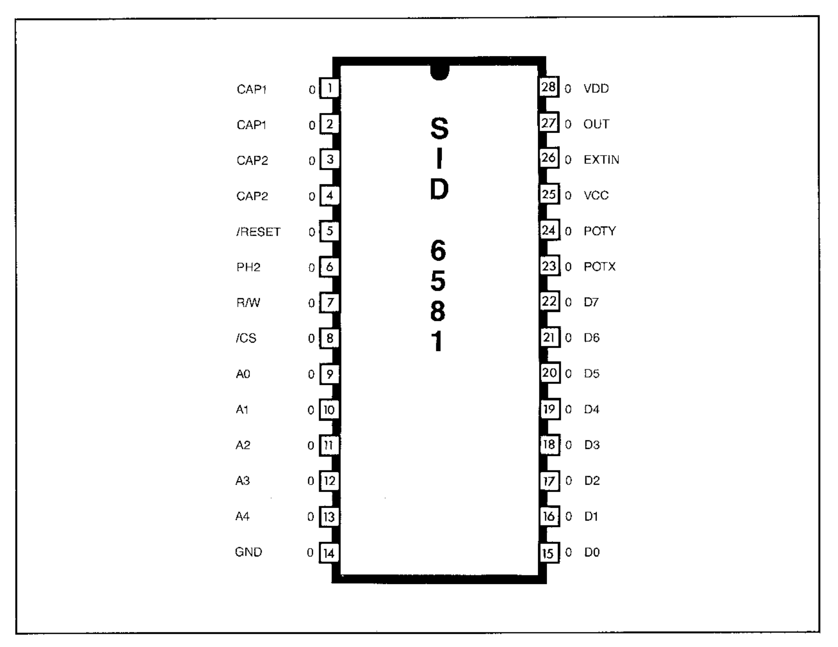
Auch hier sollen wieder einige Leitungen besonders betrachtet werden:

EXTIN (Pin 26)

Erlaubt, wie bereits erwähnt, das Einspeisen externer Tonsignale in den Baustein. Diese Leitung ist an der Audio/Video-Buchse zu finden. Das Signal kann nur über den Filter beeinflußt werden.

POTX, POTY (Pin 23, 24)

Über diese Eingänge werden Paddles abgefragt. Paddles bestehen lediglich aus einem regelbaren Widerstand. Dieser wird zwischen +5V und einem der Eingänge POTX oder POTY angeschlossen. Der SID wandelt dann den Widerstandwert in einen Zahlenwert um. Da zwei Eingänge existieren, können gleichzeitig zwei Paddles ausgewertet werden. Der C 64 erlaubt jedoch den Anschluß von zwei Paddlepaaren,



d.h. vier Paddles. Diese können jedoch nicht gleichzeitig ausgewertet werden, sondern es muß mit Hilfe eines Ein-Ausgabebausteins CIA zwischen den beiden Paaren umgeschaltet werden.

2/2.1.4

Der Video-Controller 6567 (VIC)

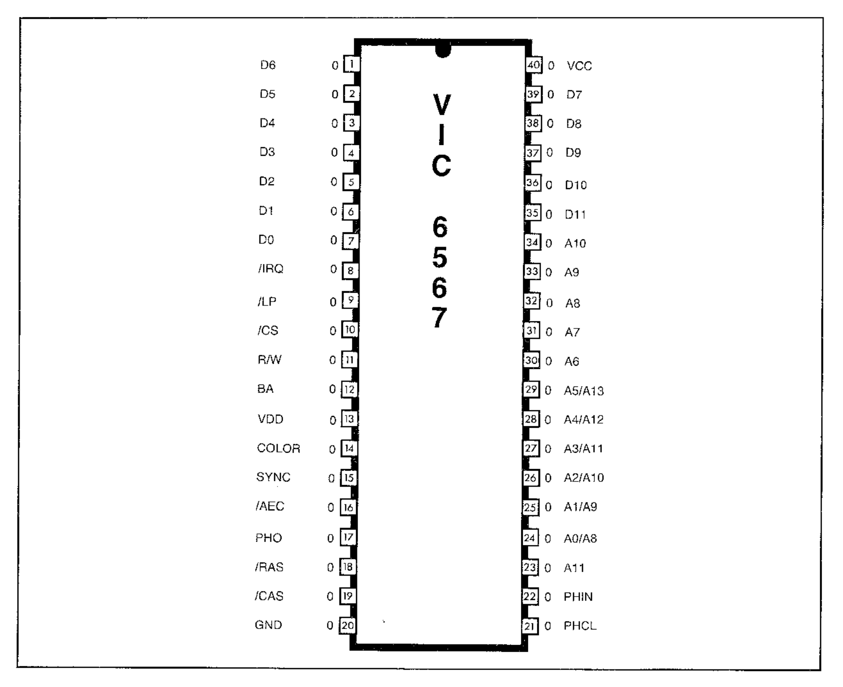
Der Video-Controller VIC übernimmt alle Arbeiten, die zur Erstellung des Bildes notwendig sind, d.h. Auswerten des Bildschirmspeichers, Farbspeichers, Zeichensatz

und Spritedaten sowie die Erzeugung sämtlicher Signale für einen Monitor. Er ist der einzige Baustein, der neben der CPU eigenständig auf den Adreßbus zugreifen kann. Ein Zugriff des Video-Controllers hat dabei sogar Vorrang vor der CPU. Da der VIC nur 14 Adreßleitungen besitzt, kann er nur einen 16 KByte Adreßbereich ansteuern. Dieser Adreßbereich kann beim C 64 aber in 16 KByte Schritten frei gewählt werden, indem die Adreßleitungen 14 und 15 von dem Ein/Ausgabebaustein CIA #2 erzeugt werden. Im Normalfall ist der Speicherbereich von 0 bis 16383 gewählt. Der VIC greift dabei immer auf das RAM zu, d.h. auch in den Bereichen, in denen die CPU auf ROM zugreifen würde.

Unabhängig von der gewählten Adreßlage liegt der Farbspeicher immer im Bereich von $D800 bis $DBFF. Dies hat den Grund darin, daß der Farbspeicher aus einem zusätzlichen RAM-Speicher besteht, der die Adreßgröße von 1 KByte besitzt, jedoch nur vier Datenleitungen enthält, was für die Farbinformation völlig ausreichend ist.

Der Zeichensatz hingegen kann verschoben werden. Im Normalfall liegt er im ROM ab $D000. Dies ist außerhalb des vom VIC zu adressierendem Speicherbereichs, kann aber durch einen Trick trotzdem angesprochen werden: Der Adreßbereich von $D000 bis $DFFF wird von der Adreßdekodierung für den VIC in die Adreßbereiche $1000 bis $1FFF und von $9000 bis $9FFF gespiegelt.

Pin	Name	Beschreibung
1-7	D6..D0	Datenbus Bit 0 bis Bit 6
8	/IRQ	Interrupt-Ausgang zum Prozessor
9	/LP	Lightpen-Anschluß
10	/CS	Bausteinauswahl
11	R/W	Lese-Schreibsignal
12	BA	Busfreigabe
13	VDD	Versorgungsspannung: +12V
14	COLOR	Ausgang Farbsignal
15	SYNC	Ausgang Synchronisationssignale
16	/AEC	0=VIC benutzt den Bus
17	PHO	Taktsignal
18	/RAS	Ansteuersignal für RAMs
19	/CAS	Ansteuersignal für RAMs
20	GND	Versorgungsspannung: Masse
21	PHCL	Eingang Farbfrequenz
22	PHIN	Eingang Dotfrequenz
23	A11	Adreßleitung
24-29	A0/A8..A5/A13	Adreßleitungen
30-34	A6..A10	Adreßleitungen
35-38	D8..D11	Datenbus für das Farbram
39	D7	Datenbus Bit 7
40	VCC	Versorgungsspannung: +5V



Einige besonders bemerkenswerte Leitungen sind im nachfolgenden näher erläutert:

/IRQ (Pin 8)

Der VIC ist in der Lage, bei bestimmten programmierbaren Ereignissen ein Interrupt-Signal zu erzeugen und an die CPU weiterzuleiten. Zu diesen Ereignissen gehört die Spritekollision mit Sprites oder dem Hintergrund, der Light-Pen sowie der Rasterzeilenvergleich. Bei dem Rasterzeilenvergleich wird der Interrupt bei dem Zeilenaufbau einer vom Programmierer festgelegten Rasterzeile ausgelöst. Dies ermöglicht viele Effekte, wie z.B. einen mehrfarbigen Rahmen oder Text- und Grafikanzeige gleichzeitig. Auch die Verdoppelung der Sprites auf 16 ist so möglich, wenn man ihre Bewegungsfreiheit auf Teilbereiche des Bildschirms beschränkt.

/LP (Pin 9)

Der LP-Eingang erlaubt den Anschluß eines Light-Pens. Das Signal kann am Joystick-Port angeschlossen werden. Im Kapitel 7/2.6 finden Sie eine Bauanleitung für einen Light-Pen sowie weitere Erklärungen.

BA (Pin 12)

Für manche Operationen benötigt der VIC den Adreßbus außergewöhnlich lange. Dies ist z.B. im Zusammenhang mit der Spritedarstellung der Fall. Dem Prozessor wird dann über die BA-Leitung mitgeteilt, daß der VIC den Bus noch benötigt.

/AEC (Pin 16)

Wirkt auf den DMA-Eingang der CPU und gibt so den Bus für den VIC frei.

/RAS und /CAS (Pin 18 und 19)

/RAS und /CAS sind Steuersignale für die Speicherbausteine. Im C 64 werden für den 64 KByte Hauptspeicher acht Speicherchips des Types 4164 verwendet. Für jede Datenleitung ist einer dieser Chips zuständig. Bei den Speicherchips handelt es sich um dynamische RAMs, welche die Information mittels kleiner Kapazitäten, die auf dem Chip integriert sind, speichern. Diese Kapazitäten sind in einer Matrix organisiert, so daß sich eine Adresse aus einer Zeilen- und einer Spaltenadresse zusammensetzt. Diese Adressen werden nacheinander an den RAM-Baustein gelegt, wobei das CAS-Signal die Spaltenauswahl und das RAS-Signal die Reihenauswahl anzeigt. Dynamische Speicher haben den Nachteil, daß die Kapazitäten mit der Zeit ihre Information verlieren. Daher muß die Information in kurzen Abständen aufgefrischt werden (Refresh genannt). Diese Arbeit wird nebenbei vollständig vom VIC erledigt.

A0/A8 bis A5/A13 (Pin 24 bis 29) sowie A6 bis A11

Die Adreßleitungen 0 bis 5 und 8 bis 13 teilen sich diese Ausgangsleitungen. Sie werden nacheinander auf die Ausgänge gelegt, da ja die Speicherbausteine die Adresse getrennt nach Zeile und Spalte verlangen. Zusammen mit den Adreßleitungen A6 und A7 ergibt sich der 16 KByte Adreßraum des VIC. Die Adreßleitungen A8 bis A11 gehen direkt an den Adreßbus.

2/2.1.5

Der Complex Interface Adapter 6526 (CIA)

Der C 64 enthält zwei IC's, über die man die CPU mit der Außenwelt in Verbindung treten lassen kann (USER-Port). Weiterhin wäre die Tastatur- und Joystickabfrage ohne diese Bausteine nicht möglich. Auch der serielle Bus, die Drucker- und Floppy-Schnittstelle, ist auf diese Bausteine angewiesen. Die beiden Complex Interface Adapter (CIA) 6526 sind diese Wunderbausteine. Sie sind die technische Weiterentwicklung der Versatile Interface Adapter (VIA) 6522, die im kleinen Bruder des C 64, dem VC 20, verwendet werden. Wie die Bezeichnung des IC's schon vermuten läßt, sind diese Bausteine für den Anschluß an die Prozessoren der 65XX-Serie entwickelt.

Da einige von Ihnen bestimmt schon schlechte Erfahrungen mit diesem Baustein gemacht haben, hier ein paar Hinweise zur Behandlung dieses IC's. Sie sollten, wenn Ihr Computer eingeschaltet ist, keine Steckmodule in den USER-Port einschieben oder abziehen, nicht das Diskettenlaufwerk an- oder abstecken und beim Anschließen der Joysticks darauf achten, daß keine Kurzschlüsse entstehen. Außerdem sollten Sie es vermeiden, mit den Kontakten des USER-Ports und der Joystickports in Berührung zu kommen, da die CIA's sehr empfindlich gegen statische Aufladungen reagieren. Diese tödliche statische Elektrizität entsteht meist durch die auftretende Reibung beim Laufen zwischen Teppichboden und Schuhsohlen (es können Spannungen bis zu 20.000 Volt auftreten).

Allgemeines über den 6526

Der 6526 hat ungeahnte Möglichkeiten, von denen nur der geringste Teil im C 64 genützt wird. Diese Geheimnisse sollen nun gelüftet werden:

— 16 einzelne programmierbare Ein-/Ausgabeleitungen
— 2 Handshake-Pins für parallele Ein-/Ausgabe
— 2 unabhängige kaskadierbare 16-Bit-Timer
— eine 24 Stunden-(AM/PM) Echtzeituhr mit programmierbarer Alarmzeit
— ein 8-Bit-Schieberegister für die serielle Ein-/Ausgabe.

Die Pinbelegung der CIA's

Die CIA 6526 besitzt ein 40-poliges Gehäuse. Dabei markiert die Einkerbung beim IC die Seite von Pin 1 und Pin 40. Betrachtet man nun das IC so, daß die Einkerbung nach oben zeigt, so ist links davon Pin 1. Man zählt nun auf der linken

Seite nach unten bis Pin 20, springt auf die rechte Seite, dort befindet sich Pin 21. Nach oben weiter gezählt endet man bei Pin 40. Diese Zählweise ist bei allen IC's genormt. Hier ist nun die Pinbelegung der CIA's:

Pin 1	Masse
Pin 2-9	Ein-/Ausgabeport A (8 Bit bidirektional)
Pin 10-17	Ein-/Ausgabeport B (8 Bit bidirektional). Die Bits 6 und 7 können zur Anzeige des Unterlaufs der beiden 16-Bit-Timer programmiert werden.
Pin 18	PC (Port Control) ist ein Ausgang, er zeigt an, daß Daten am Port B oder an beiden Ports zur Verfügung stehen.
Pin 19	TOD (Time Of Day) ist nur ein Eingang, er dient zur Triggerung der Echtzeituhr (50/60 Hertz)
Pin 20	+5V Betriebsspannung
Pin 21	IRQ (Interrupt Request) ist ein Ausgang. Dieser wird auf 0 gesetzt, wenn ein, durch das Setzen eines Bits im ICR (Interrupt Control Register), freigegebenes Interruptereignis eintritt.
Pin 22	R/W (lesen/schreiben) ist nur ein Eingang, 0 bedeutet dabei, daß der Datenbus zum Prozessor auf Eingabe geschaltet ist, daß heißt, daß der Prozessor Daten in den CIA schreiben kann. 1 bedeutet entsprechend, daß der Datenbus zum Prozessor auf Ausgabe geschaltet ist, daß heißt, daß der Prozessor Daten vom CIA empfangen kann.
Pin 23	CS (Chip Select) ist nur ein Eingang: 0=Datenbus wird übernommen; 1=Datenübergabe ist gesperrt.
Pin 24	—FLAG ist nur ein Eingang, gleiche Bedeutung wie PC.
Pin 25	02 (Systemtakt 2) ist nur ein Eingang, er ist mit dem Systemtakt des Prozessors verbunden und regelt die Kommunikation mit dem Prozessor 6510.
Pin 26-33	DB7-DB0 (Datenbus) bidirektional, direkte Verbindung mit dem Prozessor 6510.
Pin 34	RES (Reset) ist nur ein Eingang, 0=Rücksetzen der CIA in den Grundzustand.
Pin 35-38	RS3-RS0 (Register Select) ist nur ein Eingang, damit können die 16 Register der CIA angesprochen werden, aber nur wenn —CS=0 ist.
Pin 39	SP (Serial Port) bidirektional, dient zur Ein-/Ausgabe des Schieberegisters des seriellen Ports.
Pin 40	CNT (Count) bidirektional, er ist der Ein-/Ausgang des Schieberegistertaktes oder zur externen Triggerung der 16-Bit-Timer.

Die Register der CIA's

Da wir nun die Pinbelegung der CIA's kennen, wenden wir uns den einzelnen Registern zu. Durch das Memory-Mapping der Peripherie-Bausteine im C 64 ist es möglich, die CIA's genau wie Speicherplätze anzusprechen. Dabei liegen die Basisadressen der CIA's für CIA 1 bei 56320 ($DC00) und für CIA 2 bei 56576 ($DD00).

Um nun die einzelnen Registeradressen berechnen zu können, addiert man zu der Basisadresse des gewünschten CIA's die Registerzahl, so erhält man die Adresse des gewünschten Registers.

Reg 0	**PRA** (Port Register A) Zugriff: lesen/schreiben. In diesem Register kann man den Zustand der 8 Ein-/Ausgabeleitungen PA0-7 abfragen. Dabei spielt es keine Rolle, ob die I/O-Leitungen auf Ein- oder Ausgabe geschaltet sind.
Reg 1	**PRB** (Port Register B) Zugriff: lesen/schreiben. In diesem Register kann man den Zustand der 8 Ein-/Ausgabeleitungen PB0-7 abfragen. Dabei spielt es keine Rolle, ob die I/O-Leitungen auf Ein- oder Ausgabe geschaltet sind.
Reg 2	**DDRA** (Data Direction Register A) Zugriff: lesen/schreiben. Mit diesen Bits können die Datenrichtungen der Ausgabeleitungen PA0-7 festgelegt werden. Bei gesetztem Bit (1) ist die entsprechende PA-Leitung auf Ausgang und bei gelöschtem Bit (0) ist die entsprechende PA-Leitung auf Eingang geschaltet.
Reg 3	**DDRB** (Data Direction Register B) Zugriff: lesen/schreiben. Mit diesen Bits können die Datenrichtungen der Ausgabeleitungen PB0-7 festgelegt werden. Bei gesetztem Bit (1) ist die entsprechende PB-Leitung auf Ausgang und bei gelöschtem Bit (0) ist die entsprechende PB-Leitung auf Eingang geschaltet.
Reg 4	**TA LO** (Timer A LO-Byte) Zugriff: lesen. Beim Lesen dieses Registers erhält man das niederwertige Byte des 16-Bit-Timer. Dieser Timer hält beim Lesen nicht an, sondern zählt weiter. Das heißt, man liest den Momentanwert des Timers aus. Zugriff: schreiben. Beim Schreiben in dieses Register schreibt man das niederwertige Byte in den 16-Bit-Timer, von dem aus der Timer von 0 ab aufwärts zählen soll.
Reg 5	**TA HI** (Timer A HI-Byte) Zugriff: lesen. Beim Lesen dieses Registers erhält man das höherwertige Byte des 16-Bit-Timers. Dieser Timer hält beim Lesen nicht an, sondern zählt weiter. Das heißt, man liest den Momentanwert des Timers aus. Zugriff: schreiben. Beim Schreiben in dieses Registers schreibt man das höherwertige Byte in den 16-Bit-Timer, von dem aus der Timer von 0 ab aufwärts zählen soll. Ein neuer Wert wird erst in den Timer geladen, wenn laufender Zählvorgang abgeschlossen ist. Soll der Timer sofort seine Arbeit aufnehmen, so ist zusätzlich in Register 14 Bit 4 zu setzen.
Reg 6	**TB LO** (Timer B LO-Byte) Dieses Register arbeitet analog wie Register 4 bei Timer A, jedoch bezogen auf Timer B.
Reg 7	**TB HI** (Timer B HI-Byte) Dieses Register arbeitet analog wie Register 5 bei Timer A, jedoch bezogen auf Timer B.
Reg 8	**TOD** (Time Of Day **1/10 Sekunde**) Zugriff: lesen. In diesem Register werden nur die Bits 0-3 verwendet, sie geben die Zehntelsekunden der Echtzeituhr im BCD-Format (Binar Coded Decimal) an. Die Bits 4-7 sind dabei immer 0. Zugriff: schreiben **wenn in Register 15 (CRB) Bit 7 gesetzt ist.** Dann können die 1/10 Sekunden der **Alarmzeit** im BCD-Format gesetzt werden. Es ist dabei noch zu beachten, daß die Bits 4-7 auf 0 gesetzt sind. Zugriff: schreiben **wenn in Register 15 (CRB) Bit 7 nicht gesetzt ist.** Dann können die 1/10 Sekunden der **Uhrzeit** im BCD-Format gesetzt werden. Es ist dabei noch zu beachten, daß die Bits 4-7 auf 0 gesetzt sind. Außerdem wird beim Schreiben in dieses Register die Uhr wieder gestartet.

Reg 9	**TOD SEC** (Time Of Day **Sekunden**) Zugriff: lesen. In diesem Register stehen die Sekunden im BCD-Format zur Verfügung. Bit 0-3, dort stehen die Einersekunden (BCD) Bit 4-6, dort stehen die Zehnersekunden (BCD) Bit 7 ist immer 0 Zugriff: schreiben **wenn in Register 15 (CRB) Bit 7 gesetzt ist.** Dann können die Sekunden der **Alarmzeit** im BCD-Format gesetzt werden. Zugriff: schreiben **wenn in Register 15 (CRB) Bit 7 nicht gesetzt ist.** Dann können die Sekunden der **Uhrzeit** im BCD-Format gesetzt werden.
Reg 10	**TOD MIN** (Time Of Day **Minuten**) Dieses Register arbeitet analog wie Register 9, jedoch bezogen auf Minuten.
Reg 11	**TOD** (Time Of Day **Stunden**) Zugriff: lesen. In diesem Register stehen die Stunden im BCD-Format zur Verfügung. Bit 0-3, dort stehen die Einerstunden (BCD) Bit 4, dort stehen die Zehnerstunden (BCD) Bit 5, 6, sind immer 0 Bit 7, dort steht die Tageshälfte. Ist Bit 7 gelöscht (0), signalisiert dieses die Tageshälfte AM (vormittags). Ist Bit 7 gesetzt (1), signalisiert dieses die Tageshälfte PM (nachmittags). Zugriff: schreiben **wenn in Register 15 (CRB) Bit 7 gesetzt ist.** Dann können die Stunden der **Alarmzeit** im BCD-Format gesetzt werden. Zugriff: schreiben **wenn in Register 15 (CRB) Bit 7 nicht gesetzt ist.** Dann können die Stunden der **Uhrzeit** im BCD-Format gesetzt werden. Beim Schreiben in dieses Register wird die Uhr angehalten. Sie startet erst wieder, wenn ein Schreibzugriff auf das Register 8 erfolgte.
Reg 12	**SDR** (Serial Data Register) Zugriff: lesen/schreiben. Dieses Register wirkt auf die Anschlüsse SP und CNT. Es ist ein Schieberegister zum Senden oder Empfangen von seriellen Daten. Die Datenrichtung wird von Bit 6 in Register 14 bestimmt. Eingabemodus: Ein Bit wird in das Schieberegister übernommen, wenn am Anschluß CNT eine steigende Flanke auftritt. Nachdem acht Bit übernommen wurden, wird das entsprechende Bit im Register 13 (ICR) gesetzt. Ausgabemodus: Die Schieberate wird durch den Timer A bestimmt. Die Schiebeimpulse erscheinen am Anschluß CNT. Ist das Schieberegister leer, so wird ebenfalls das Bit im Interrupt-Kontrollregister gesetzt, es sei denn, daß vor Beendigung des Schiebevorgangs ein neuer Wert in das Schieberegister geschrieben wurde.
Reg 13	**ICR** (Interrupt Control Register) Zugriff: lesen (INT-Data). **Achtung!** Beim Lesen des INT-Data-Registers werden alle Bits des INT-Data-Registers auf Null gesetzt. Nun, was verbirgt dieses Register. Es zeigt die einzelnen Zustände, die im CIA auftreten können, an. Das Register kann nun so programmiert werden, daß beim Auftreten bestimmter Zustände ein Interrupt an die CPU gesendet wird. Hier nun die einzelnen Bits dieses Registers und ihre Funktionen.
Bit 0	ist Bit 0 gesetzt (1), dann tritt ein Unterlauf an Timer A auf.
Bit 1	ist Bit 1 gesetzt (1), dann tritt ein Unterlauf an Timer B auf.
Bit 2	ist Bit 2 gesetzt (1), dann stimmt die Uhrzeit mit der Alarmzeit überein.
Bit 3	wird gesetzt, wenn das SDR (Serial Data Register) je nach Betriebsart voll oder leer ist.
Bit 4	wird gesetzt, wenn am Eingang von Pin 24 (FLAG) eine negative Flanke auftritt.
Bit 5	ist immer 0 (ohne Bedeutung).

Bit 6	ist immer 0 (ohne Bedeutung).
Bit 7	Die Daten, die im INT-Mask-Register abgelegt werden, bestimmen, bei welchen Ereignissen ein Interrupt ausgelöst werden soll. Ist ein Bit im INT-Mask-Register gesetzt (1) und es wird das gleiche Bit vom CIA im INT-Data-Register gesetzt, dann wird Bit 7 im INT-Data-Register gesetzt und gleichzeitig ein Interrupt an die CPU über Pin 21 (-IRQ) gesendet. Zugriff: schreiben (INT-Mask). **Achtung!** Beim Schreiben in das INT-Mask-Register werden alle Bits des INT-Data-Registers auf Null gesetzt. Um Daten in das INT-Mask-Register zu schreiben, muß man sich vorher überlegt haben, ob man ein Bit löschen oder ob man es setzen möchte. Dafür ist im INT-Mask-Register das Bit 7 zuständig. Die anderen Bits sind identisch mit dem INT-Data-Register. Möchte man nun ein Bit löschen, so muß gleichzeitig das Bit 7 gelöscht und das zu löschende Bit gesetzt werden. Soll ein Bit gesetzt werden, so muß Bit 7 gesetzt und gleichzeitig das zu setzende Bit gesetzt werden.
Reg 14	**CRA** (Control Register A) Zugriff: lesen/schreiben. Mit diesem Register werden die Betriebszustände des CIA's bestimmt. Es kommt in diesem Register wieder auf die einzelnen Bits an.
Bit 0	Ist Bit 0 gesetzt, dann startet Timer A. Ist Bit 0 gelöscht, dann stoppt Timer A.
Bit 1	Ist Bit 1 gesetzt, so wird jeder Unterlauf von Timer A an Pin 16 (PB6) signalisiert. Dieses Signal hat Vorrang vor der Festlegung im DDRB. Wie das Signal nun aussieht, wird in Bit 2 festgelegt.
Bit 2	Ist Bit 2 gesetzt, so wird ein Unterlauf von Timer A durch ein Umkippen von PB6 auf das jeweils andere Potential signalisiert. Ist Bit 2 gelöscht, so wird ein Unterlauf von Timer A durch einen positiven Impuls in der Länge eines Systemtaktes (ca. 1 Microsekunde) angezeigt. Ist Bit 3 gesetzt, dann zählt Timer A nur einmal vom programmierten Wert auf Null und hält dann an (One Shot Mode).
Bit 3	Ist Bit 3 gelöscht, dann zählt Timer A ständig vom programmierten Wert nach Null. Das bedeutet, sobald der Timer den Wert Null erreicht hat, beginnt er wieder beim anfangs programmiertem Wert auf Null herunterzuzählen (Continious Mode).
Bit 4	Ist 4 Bit gesetzt, dann ist es möglich, den Timer zu jedem beliebigen Zeitpunkt mit einem neuen Wert zu laden.
Bit 5	Ist Bit 5 gesetzt, dann ist es möglich, den Timer A extern zu triggern. Es werden dabei die positiven Flanken an Pin 40 (CNT) gezählt. Ist Bit 5 gelöscht, dann wird der Systemtakt gezählt.
Bit 6	Ist Bit 6 gesetzt, dann werden aus dem Schieberegister Daten herausgeschoben. Es arbeitet als Ausgang. Ist das Bit gelöscht, dann arbeitet das Schieberegister als Eingang.
Bit 7	Ist Bit 7 gesetzt, so arbeitet die Echtzeituhr mit der Netzfrequenz von 50 Hz. Ist das Bit 7 gelöscht, so arbeitet die Echtzeituhr mit der Frequenz von 60 Hz (wie in den USA üblich ist).
Reg 15	**CRB** (Control Register B) Zugriff: lesen/schreiben.
Bit 0	Ist Bit 0 gesetzt, dann startet Timer B. Ist Bit 0 gelöscht, dann stoppt Timer B.
Bit 1	Ist Bit 1 gesetzt, so wird jeder Unterlauf von Timer B am Pin 17 (PB7) signalisiert. Dieses Signal hat Vorrang vor der Festlegung im DDRB. Wie das Signal nun aussieht, wird in Bit 2 festgelegt.
Bit 2	Ist Bit 2 gesetzt, so wird ein Unterlauf von Timer B durch ein Umkippen von PB7 auf das jeweils andere Potential signalisiert. Ist Bit 2 gelöscht, wo wird ein Unterlauf von Timer B durch einen positiven Impuls in der Länge eines Systemtaktes angezeigt.

Bit 3	Ist Bit 3 gesetzt, dann zählt Timer B nur einmal vom programmierten Wert auf Null und hält dann an (One Shot Mode). Ist Bit 3 gelöscht, dann zählt Timer B ständig vom programmierten Wert auf Null herunter (Continious Mode).
Bit 4	Ist Bit 4 gesetzt, dann ist es möglich in den Timer zu jedem beliebigen Zeitpunkt mit einem neuen Wert zu laden und sofort zu starten.
Bit 5-6	Mit diesen beiden Bits wird die Betriebsart von Timer B bestimmt. Die Betriebsart des Timers ist von **beiden Bits** abhängig. Es müssen deshalb immer beide Bits abgefragt werden, um zu wissen, wie Timer B arbeitet. **Bit 5 gelöscht/Bit 6 gelöscht** Der Timer wird vom Systemtakt getriggert. **Bit 5 gelöscht/Bit 6 gesetzt** Der Timer wird von positiven Flanken, die an Pin 40 (CNT) auftreten, getriggert. **Bit 5 gesetzt/Bit 6 gelöscht** Der Timer B wird von Timer A getriggert. Jeder Unterlauf von Timer A ist dabei ein Triggerimpuls. **Bit 5 gesetzt/Bit 6 gesetzt** Timer B wird von Timer A nur dann getriggert, wenn an Pin 40 (CNT) ein High-Pegel anliegt (CNT muß 1 sein). Das bedeutet, die Unterläufe von Timer A werden nur dann an Timer B weitergegeben, wenn an Pin 40 (CNT) ein High-Pegel anliegt.
Bit 7	Ist Bit 7 gesetzt, so kann man die Uhrzeit von der Echtzeituhr in den Registern 8-11 (TOD . . .) einstellen.

Die Aufgaben der CIA's im C 64

Die Aufgaben der E/A-Ports von CIA 1:

Die Tastaturabfrage:
Die Tastatur ist als eine 8x8-Matrix aufgebaut (siehe Bild 2/2.1.5-1). Diese Matrix wird von den zwei 8-Bit-Ports des CIA 1 abgefragt. An Port A von CIA 1 werden die Zeilen des Tastenfeldes und an Port B die Spalten abgefragt. Da das Tastenfeld des C 64 aber 66 Tasten enthält, die sich jedoch mit einer 8*8-Matrix nur 64 Tasten abfragen lassen, werden sich bestimmt schon manche von Ihnen gefragt haben, wie das realisiert wird. Die Lösung ist sehr einfach. Die SHIFT-LOCK-TASTE wird parallel zu einer SHIFT-Taste angeschlossen. Nun bleibt noch eine Taste übrig, die RESTORE-Taste. Sie hat nur die Aufgabe, einen NMI auszulösen. Die RESTORE-Taste wird deshalb direkt über ein Timer-IC (NE 556), das sich im Steckplatz U20 auf der Rechnerplatine befindet, angeschlossen. Beim Betätigen dieser Taste gibt der Timer einen kurzen Impuls an die NMI-Leitung der CPU weiter. Dieser NMI-Anschluß wird bei der Beschreibung des Prozessors noch genauer behandelt. Damit sind alle 66 Tasten untergebracht.

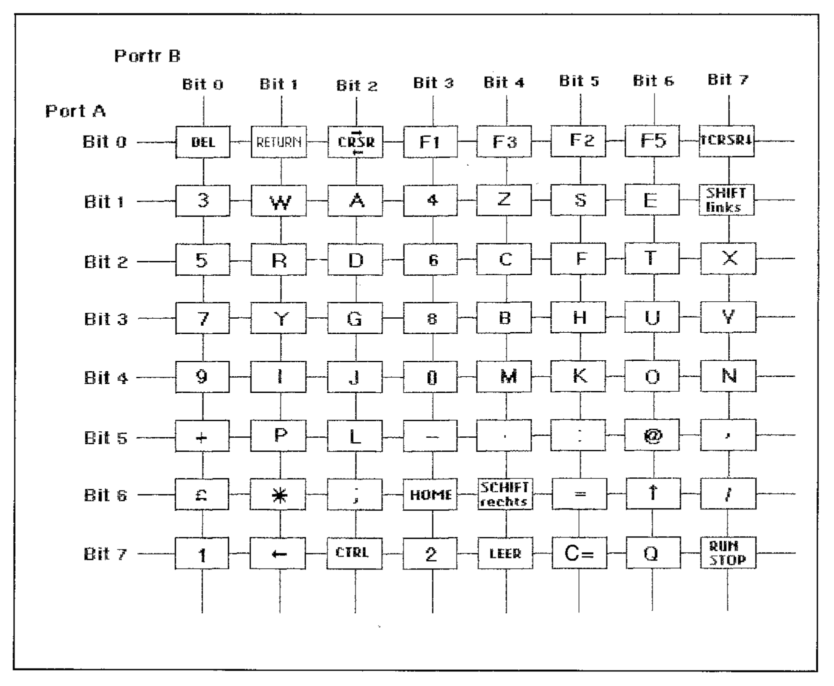


Bild 2/2.1.5-1: Der Aufbau der Tastatur-Matrix

Was passiert, wenn nun eine Taste innerhalb der 8*8-Matrix gedrückt wird?

Zuvor noch ein Hinweis, der zum besseren Verstehen der Tastaturabfrage wichtig ist. Wenn am CIA eine Port-Leitung auf Eingang geschaltet ist und diese Leitung nicht belegt wird, so interpretiert der CIA-Baustein an diesem Eingang einen High-Pegel (logisch 1).

Die Ein-/Ausgabeleitungen (PA0 bis PA7 und PB0 bis PB7) werden beim Einschalten Ihres Rechners vom Betriebssystem programmiert. Der Port A (PA0 bis PA7) arbeitet als Ausgang und der Port B (PB0 bis PB7) arbeitet als Eingang. Da diese Eingänge ja nicht beschaltet sind (das heißt, es liegt kein definierter Zustand logisch 0 oder 1 an), bedeutet das für den CIA, daß nun an allen Eingängen ein High-Pegel anliegt. Tritt nun ein Interrupt auf (normal jede 1/60 Sekunde), ausgelöst vom Timer des CIA 1, so fragt das Betriebssystem unter anderem auch die Tastatur ab. Bei dieser Abfrage geschieht nun folgendes:

Es wird der Port A (PA0 bis PA7 alle Leitungen gleichzeitig) für einen kurzen Moment auf Low-Pegel gesetzt. Solange nun dieser Port einen Low-Pegel führt, wird Port B der Reihe nach von PA0 bis PA7 abgefragt. Wenn nun eine Taste innerhalb der 8*8-Matrix gedrückt wird, so liegt an einem dieser Port-Leitungen ein Low-Pegel an. Sobald dieser Low-Pegel an einer der Port-Leitungen erkannt wird, weiß die CPU, daß eine Taste gedrückt ist. Sie muß jetzt nur noch herausfinden, welche Taste in der Tastaturmatrix gedrückt ist. Dazu schaltet die CPU der Reihe nach die einzelnen Port-Leitungen von Port A (PA0 bis PA7) auf Low-Pegel und überprüft dabei, wann an einer Portleitung von Port B ein Low-Signal auftritt. Ist die Leitung gefunden, steht die genaue Position der gedrückten Taste in der 8*8-Matrix fest. Diese Bitkombination kann das Betriebssystem jetzt auswerten. Sie können die einzelnen Positionen der Tasten aus der Matrix der Zeichnung 2/2.1.5-1 entnehmen.

Die Joystick-Abfrage

Eine weitere Aufgabe der Ein-Ausgabe-Ports von CIA 1 ist die Joystick-Abfrage an die Control-Ports 1 und 2. Es werden für die Joystick-Abfrage jeweils 6 Leitungen pro Joystick und Controlport genutzt. Davon werden vier Leitungen für die vier Hauptrichtungen oben, unten, links und rechts genutzt. Die fünfte Leitung signalisiert den Feuerknopf und die sechste Leitung ist der Masse-Anschluß. Hier nun die Pinbelegung der für die Joystick-Abfrage benötigten Anschlüsse:

Von Control-Port 1 werden folgende Anschlüsse benötigt:

	Pin vom Stecker	Signal	CIA Pin
oben:	1	JOY A0	PB0 (10)
unten:	2	JOY A1	PB1 (11)
links:	3	JOY A2	PB2 (12)
rechts:	4	JOY A3	PB3 (13)
Feuer:	6	Button A/LP*	PB4 (14)
Masse:	8	GND	— (1)

* Dieser Eingang wird auch verwendet um einen Light-Pen anzuschließen (wie schon beschrieben).

Von Control-Port 2 werden folgende Anschlüsse benötigt:

	Pin vom Stecker	Signal	CIA Pin
oben:	1	JOY B0	PA0 (2)
unten:	2	JOY B1	PA1 (3)
links:	3	JOY B2	PA2 (4)
rechts:	4	JOY B3	PA3 (5)
Feuer:	6	Button B	PA4 (6)
Masse:	8	GND	— (1)

Die Anschlußbelegung der Control-Ports können Sie aus dem 2. Teil des Kapitel 2/1.1.1.2 über die Control-Ports entnehmen.

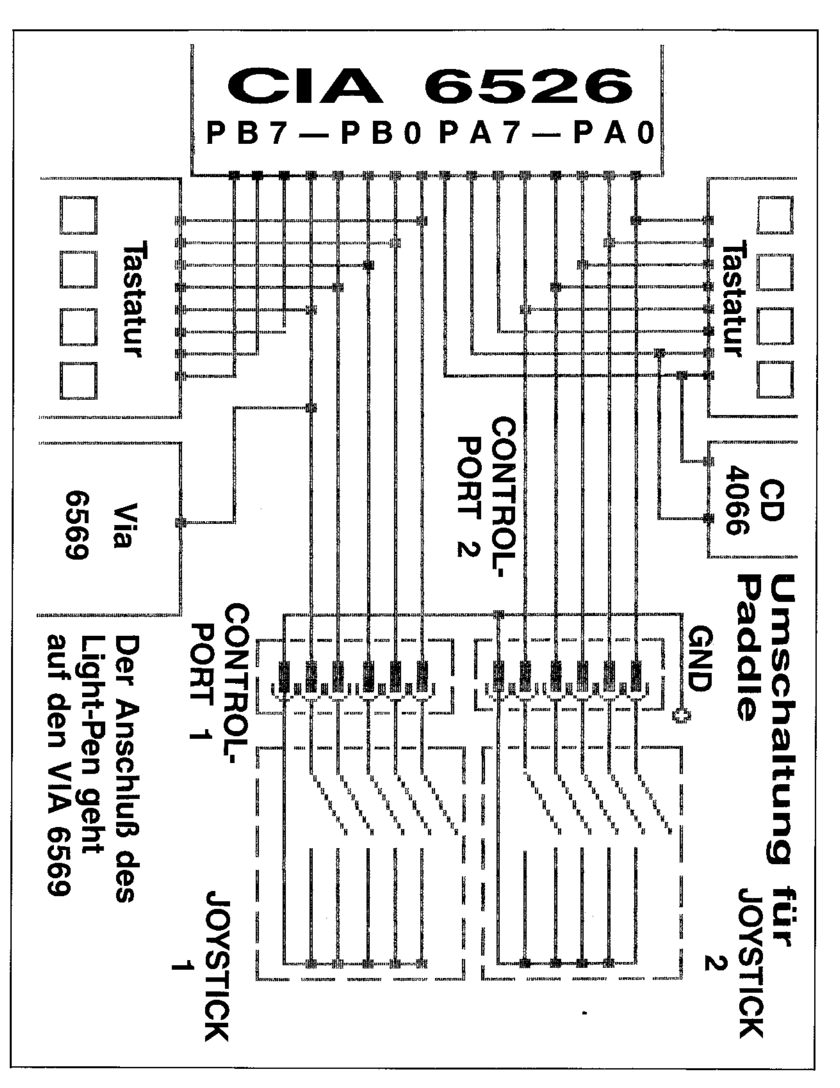

CIA 6526
PB7—PB0 PA7—PA0
Tastatur
Tastatur
Via 6569
CONTROL-PORT 2
CD 4066
Umschaltung für Paddle
GND
CONTROL-PORT 1
Der Anschluß des Light-Pen geht auf den VIA 6569
JOYSTICK 1
JOYSTICK 2

Bild 2.1.5-2

Aufbau eines Joysticks:

Im Inneren des Joysticks befinden sich fünf Schaltkontakte (vier Richtungen und der Feuerknopf), die im Ruhestand geöffnet sind. Wird der Joystick nun bewegt, so wird ein Schaltkontakt betätigt. Dieser Schaltkontakt legt nun an den entsprechenden Port des CIA 1 ein Low-Signal. Kommt am CIA dieses Signal an, so wird es verarbeitet und führt je nach Programmierung einen Befehl aus. Zum besseren Verstehen betrachten Sie bitte den Schaltplan (s. Seite 9).

Die Aufgaben der E/A-Ports von CIA 2 (NMI-CIA):

Die Ein-/Ausgabe-Ports von CIA 2 werden hauptsächlich für Datenübertragungen genutzt.

Von Port A sind nun folgende Leitungen belegt:

Bit 0 und Bit 1 dienen zur Festlegung der Video-Bank für den VIC-II-Chip. Das heißt, mit diesen beiden Leitungen wird der Speicherbereich, der vom VIC-II-Chip genutzt werden kann, festlegt.

Bit 2 wird für die Datenausgabe bei einer angeschlossenen RS-232 Schnittstelle benötigt (TXD).

Die übrigen Bits werden für die Übertragung der seriellen Schnittstelle benötigt.

Bit 3:	ANT OUT	(Attention-Leitung)
Bit 4:	CLOCK OUT	(Taktleitung Ausgang)
Bit 5:	SERIAL OUT	(Datenleitung Ausgang)
Bit 6:	CLOCK IN	(Taktleitung Eingang)
Bit 7:	SERIAL IN	(Datenleitung Eingang)

Port B wird im Normalbetrieb nicht benutzt. Die Leitungen von Port B sind vollständig an den USER-Port herausgeführt und stehen dort zur freien Verfügung. Wird am USER-Port eine RS-232-Schnittstelle angeschlossen, sind die Port-Leitungen von Port B wie folgt belegt:

Bit 0:	RXD	(Receive Data)
Bit 1:	RTS	(Request To Send)
Bit 2:	DTR	(Data Terminal Ready)
Bit 3:	RI	(Ring Indicator)
Bit 4:	DCD	(Data Carrier Detect)
Bit 5:		unbenutzt
Bit 6:	CTS	(Clear To Send)
Bit 7:	DSR	(Data Set Ready)

2/2.1.6

Das logische Progammierfeld (PLA)

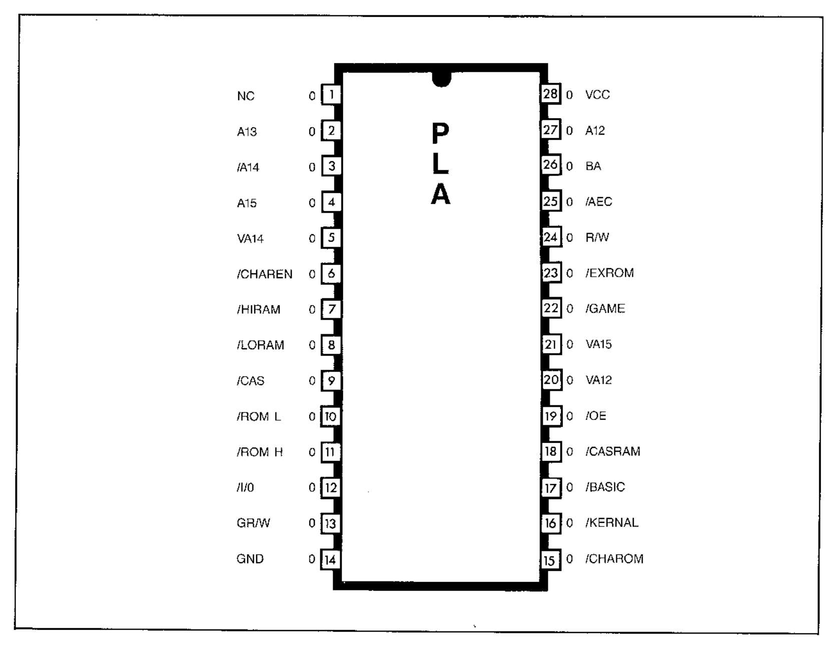
Das PLA (Programable Logic Array, engl. programmierbares Logikfeld) ist ein spezieller IC, der, im Fall des C 64, über 16 Eingabeleitungen und 8 Ausgabeleitungen verfügt. Intern besitzt er eine Vielzahl möglicher logischer Verknüpfungen, um die Ausgangssignale aus den Eingangssignalen zu erzeugen. Der IC im C 64 wurde so programmiert, daß er den wesentlichen Teil der Adreßdekodierung übernimmt. Im folgenden ist die Anschlußbelegung im C 64 wiedergegeben:

Pin	Name	Beschreibung
1	NC	Nicht benutzt
2-4	A13..A15	Adreßleitungen 13 bis 15
5	VA14	Adreßleitung 14 des Video-Bereichs
6	CHAREN	CHAREN vom Prozessorport
7	HIRAM	HIREM vom Prozessorport
8	LORAM	LORAM vom Prozessorport
9	CAS	Signal vom VIC zur RAM-Steuerung
10	ROM L	Signal zum Modulport
11	ROM H	Signal zum Modulport
12	I/O	aktiviert Adreßdecoder für I/O-Bereich
13	GR/W	R/W-Signal für das Farbram
14	GND	Versorgungsspannung: Masse
15	CHAROM	Auswahlsignal Zeichensatz-ROM
16	KERNAL	Auswahlsignal Betriebssystem-ROM
17	BASIC	Auswahlsignal Basic-ROM
18	CASRAM	CAS-Signal für RAMs
19	OE	Output-Enable-Signal: aktiviert das PLA. Im C 64 ist das PLA immer aktiviert.
20-21	VA12..VA13	Adreßleistungen 12-13 des Video-Bereichs
22	GAME	Eingangssignal vom Modulport
23	EXROM	Eingangssignal vom Modulport
24	R/W	R/W-Signal vom Prozessor
25	AEC	AEC-Signal vom VIC
26	BA	BA-Signal vom VIC
27	A12	Adreßleitung 12
28	VCC	Versorgungsspannung: +5V

Um die vielen Steuermöglichkeiten des PLAs zu verstehen, sind hier die wichtigsten noch einmal kurz beschrieben:

/CHAREN, /HIRAM und /LORAM (Pin 6, 7 und 8)

sind die drei Speicherverwaltungssignale, die vom Prozessorport kommen. Mit Hilfe dieser drei Signale entscheidet das PLA, welche Bausteine in welchen Speicher-



bereich eingeblendet werden. Da die Auswirkungen bereits im Kapitel 3/1.1 ausführlich dargestellt sind, soll hier nicht weiter darauf eingegangen werden.

/CHAROM, /KERNAL und /BASIC (Pin 15, 16 und 17)

sind die Bausteinauswahlsignale für die drei ROMs des C 64. Das PLA steuert über diese Signale den Zeichensatz (CHAROM), das Betriebssystem (KERNAL) sowie den Basic-Interpreter an (BASIC).

/GAME und /EXROM (Pin 22 und 23)

EXROM und GAME sind Signale, die vom Modulport kommen. Mit Hilfe dieser Signale wird ein eingestecktes Modul erkannt. Die folgende Tabelle gibt kurz die sinnvollen Auswirkungen der Signale wieder:

2/2.1.6

Das logische Progammierfeld (PLA)

Das PLA (Programable Logic Array, engl. programmierbares Logikfeld) ist ein spezieller IC, der, im Fall des C 64, über 16 Eingabeleitungen und 8 Ausgabeleitungen verfügt. Intern besitzt er eine Vielzahl möglicher logischer Verknüpfungen, um die Ausgangssignale aus den Eingangssignalen zu erzeugen. Der IC im C 64 wurde so programmiert, daß er den wesentlichen Teil der Adreßdekodierung übernimmt. Im folgenden ist die Anschlußbelegung im C 64 wiedergegeben:

Pin	Name	Beschreibung
1	NC	Nicht benutzt
2-4	A13..A15	Adreßleitungen 13 bis 15
5	VA14	Adreßleitung 14 des Video-Bereichs
6	CHAREN	CHAREN vom Prozessorport
7	HIRAM	HIREM vom Prozessorport
8	LORAM	LORAM vom Prozessorport
9	CAS	Signal vom VIC zur RAM-Steuerung
10	ROM L	Signal zum Modulport
11	ROM H	Signal zum Modulport
12	I/O	aktiviert Adreßdecoder für I/O-Bereich
13	GR/W	R/W-Signal für das Farbram
14	GND	Versorgungsspannung: Masse
15	CHAROM	Auswahlsignal Zeichensatz-ROM
16	KERNAL	Auswahlsignal Betriebssystem-ROM
17	BASIC	Auswahlsignal Basic-ROM
18	CASRAM	CAS-Signal für RAMs
19	OE	Output-Enable-Signal: aktiviert das PLA. Im C 64 ist das PLA immer aktiviert.
20-21	VA12..VA13	Adreßleistungen 12-13 des Video-Bereichs
22	GAME	Eingangssignal vom Modulport
23	EXROM	Eingangssignal vom Modulport
24	R/W	R/W-Signal vom Prozessor
25	AEC	AEC-Signal vom VIC
26	BA	BA-Signal vom VIC
27	A12	Adreßleitung 12
28	VCC	Versorgungsspannung: +5V

Um die vielen Steuermöglichkeiten des PLAs zu verstehen, sind hier die wichtigsten noch einmal kurz beschrieben:

/CHAREN, /HIRAM und /LORAM (Pin 6, 7 und 8)

sind die drei Speicherverwaltungssignale, die vom Prozessorport kommen. Mit Hilfe dieser drei Signale entscheidet das PLA, welche Bausteine in welchen Speicher-

LORAM	GAME	EXROM	AUSWIRKUNG
1	1	1	keine
1	1	0	8K ROM-Modul von $8000 bis $9FFF
1	0	0	16K ROM-Modul von $8000 bis $BFFF
0	0	0	8K ROM-Modul von $A000 bis $BFFF

/ROM L und /ROM H (Pin 10 und 11)

ROM L und ROM H sind die Bausteinauswahlsignale für die zwei ROM-Bausteine des Modulports. ROM L steuert dabei den Baustein für den Speicherbereich von $8000 bis $9FFFF und ROM H den Baustein für den Bereich von $A000 bis $BFFF.

2/2.2

Interner Aufbau des C 128 PC

Nach dem Aufbau des C 64 wollen wir natürlich auch den des C 128 besprechen. Wir beginnen mit einer allgemeinen Einführung, die auch ein Blockschaltbild der vorhandenen Bausteine beinhaltet. Es folgt die Beschreibung der einzelnen Bausteine, wie des Prozessors 8502, des 80-Zeichen-Videokontrollers (VDC), des Z 80 A-Prozessors, der Speicherverwaltung MMU und des logischen Programmierfeldes (PLA). Außerdem wird noch der Videokontroller für den 40-Zeichen-Monitor beschrieben, da sich beim 8564 einige Änderungen gegenüber dem 6567 im C 64 ergeben haben.

Den Abschluß des Kapitels über den internen Aufbau des C 128 bildet eine Darstellung der Busleitungen sowie eine Zusammenfassung, die insbesondere auf das Zusammenwirken der einzelnen Bausteine eingeht.

2/2.2.1

Einführung mit Blockschaltbild

(Autor: Adrian Naftanail)

Obwohl der C 128 meistens als Nachfolger des legendären C 64 vorgestellt wird, ist er eigentlich viel mehr als eine einseitige Weiterentwicklung seines Vorgängers. Er besitzt nämlich die Fähigkeiten von drei Computern in einem. Auf Benutzerebene schlägt sich diese Tatsache im Vorhandensein von drei Arbeitsmodi nieder, und zwar:

— der C 64-Modus, 100%ig kompatibel mit dem C 64, da sein Vorgänger intern durch die Hardware simuliert wird. Es stimmen also auch alle Systemadressen überein, und die Wirkungen der verschiedenen POKE's sind somit genau dieselben.

— der C 128- oder eigene Modus. Dieser Modus stellt eine Erweiterung der Fähigkeiten des C 64 bis auf die Ebene eines Personal-Computers mit professioneller 80-Zeichen-Ausgabe und doppelter Arbeitsgeschwindigkeit dar. Die eingebaute Basic-Version 7.0 erfüllt viele Wünsche die noch offen standen, und die hervorragenden Möglichkeiten im Bereich Grafik und Sound können auch ohne umständliche POKE's oder schreiben von umfangreichen Programmteilen zur Geltung gebracht werden.

— Der CPM-Modus verwandelt schließlich den C 128 in ein leistungsfähiges Werkzeug unter dem Betriebssystem CPM Version 3.0 im 2-MHz-Takt, und erweitert somit beträchtlich das ohnehin schon sehr umfangreiche Angebot existenter Software.

Für den interessierten Computer-Fan und den fortgeschrittenen Benutzer bleibt das Vorhandensein der drei Betriebsmodi nicht nur ein äußerliches Merkmal, sondern spiegelt sich unübersehbar in der Hardware des C 128 wieder. Es ist sozusagen das Zusammenleben von drei Computern in einem Gehäuse, wobei allerdings sehr viele Bausteine zur gemeinsamen Benutzung in allen drei Systemen bereit stehen, andere jedoch ausgeschaltet oder nur mit beschränkter Funktion benutzt werden.

Um den internen Aufbau und die Funktionsweise leichter verstehen zu können, stellen wir das Blockschaltbild des Computers dar (Bild 2/2.2.1-1). In den nächsten Kapiteln folgt eine genauere Beschreibung der einzelnen Bauteile, und im letzten Kapitel eine Zusammenfassung der Beziehungen dieser Abschnitte zueinander, sofern dies nicht notwendigerweise im Zusammenhang mit der Betrachtung der Bausteine in den vorangegangenen Kapiteln bereits geschehen ist.

Bevor wir mit dem Kommentar des Blockschaltbildes fortfahren, betrachten wir nochmals global die für den Benutzer interessanten Eigenschaften des C 128. Der C 64-Modus simuliert absolut seinen Vorgänger, kann aber nicht die weiteren Möglichkeiten des Systems ausschöpfen. Für den Betrieb im eigenen Modus wurde der Commodore-Kernal auf eine kompatible Weise erweitert. Es stehen 128KB RAM für Benutzer und System zur Verfügung, mit eingebauter Basic-Version 7.0. Intern müssen die 128KB RAM allerdings in Bereiche (sogenannte Banks) aufgeteilt werden, da der Prozessor über seinen Adreßbus höchstens 64KB adressieren kann. Stack und Zero-Page können verlegt werden, was manche Umständlichkeiten bei der Software-Erstellung erspart. Das Standard-ROM des Systems beträgt 48KB. Dazu kommen noch 32KB ROM für interne Funktionen und die Anschlußmöglichkeit einer externen 32KB ROM-Erweiterung. Der Z80-Coprozessor kann die 128KB RAM sowie die Kernal-Utilities voll benutzen.

Die zum C 64 zusätzlich eingebauten Funktionstasten sind vom Benutzer frei programmierbar. Die bekannten Ein- und Ausgabe-Möglichkeiten werden mit einer zweifach schnelleren Monitor-Ausgabe, einem schnelleren Interface zum Diskettenlaufwerk und einem Audio-Eingang des SID-Chips zum Mischen von externen Audio-Signalen verbessert.

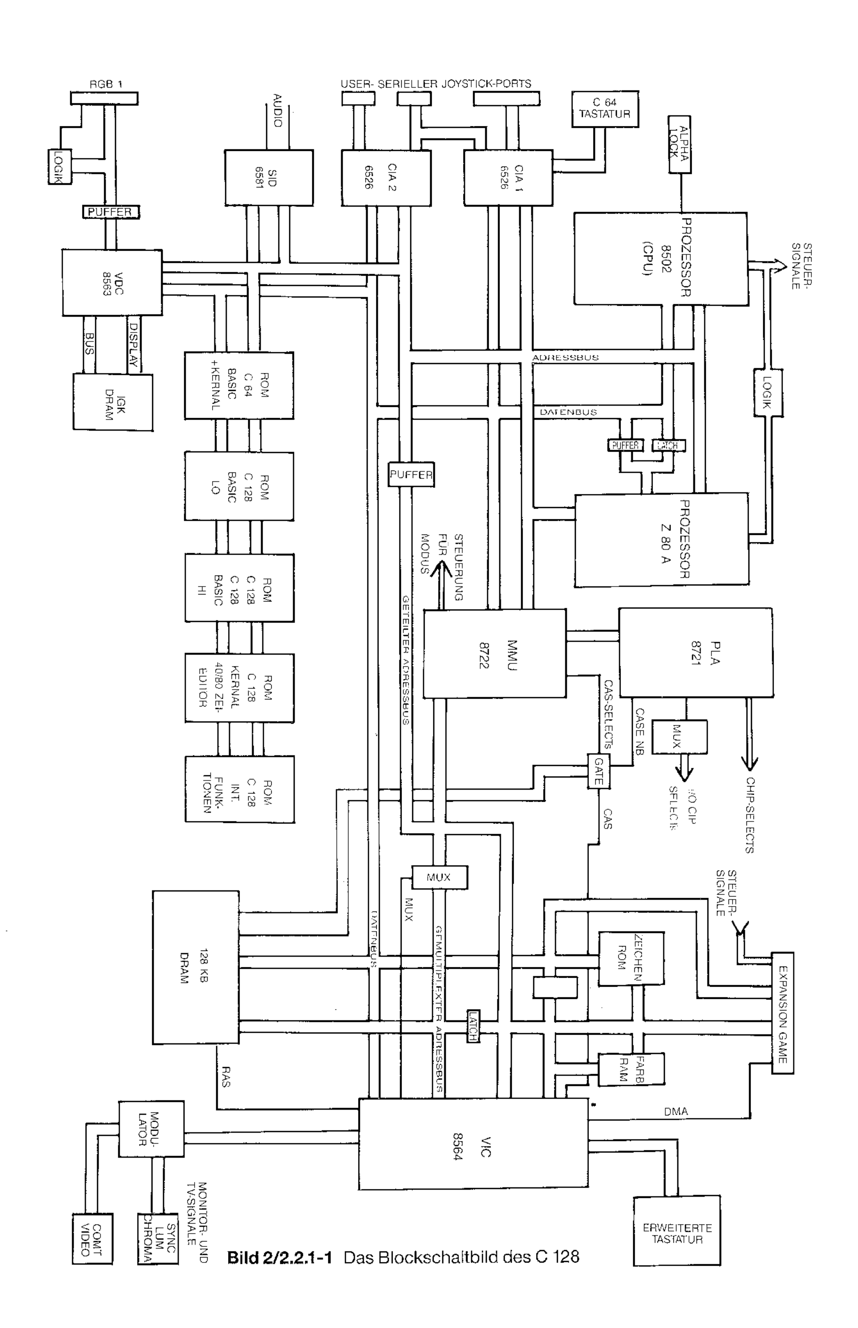


Bild 2/2.2.1-1 Das Blockschaltbild des C 128

Im folgenden werden wir nun anhand des Blockschaltbildes 2/2.2.1-1 die internen Zusammenhänge etwas näher betrachten. Damit die Verständlichkeit nicht durch nebensächliche Klammern erschwert wird, werden Bausteine und Signale mit dem Kurznamen erwähnt. Der Sinn der Bezeichnungen kann dann aus den folgenden Kapiteln entnommen werden.

Der Kommunikation des Prozessors mit den untergeordneten Bau-Elementen steht im C 128 eine sogenannte Teilhaberbus-Struktur zu Grunde. Dies erlaubt geteilten Zugriff auf das Zeichen-ROM, Farb-RAM und System-RAM sowohl von Seiten des Prozessors, wie auch des 8564-VIC Video-Controllers ohne das Zwischenschalten eines Interfaces oder die Benutzung eines Coprozessors (wie im C 64). Damit dies möglich ist, muß das RAM schnell genug sein, um in einem halben Maschinenzyklus die geforderten Daten zur Verfügung stellen zu können, und der Adreßbus wird in einen geteilten und in einen nicht geteilten Bereich gespaltet.

Alle normalen 65xx-Ein-/Ausgabeteile bleiben auf dem nichtgeteilten Adreßbus, während der VIC-Chip und seine unterstützenden Chips auf dem geteilten Bereich arbeiten. Der VIC-Chip schleust die Prozessoradressen auf den geteilten Busbereich mittels seiner AEC-Leitung. Die Kommunikation des Prozessors mit der Ein- und Ausgabe und dem ROM wird von der PLA durch Chip-select Signale geregelt.

Der Zugriff des Prozessors auf das RAM wird von der MMU mit der übersandten Adresse, der Ausgangsadresse (stets als übersetzte Adresse angesprochen), sowie des Eintrages im MMU-Konfigurationsregister und RAM-Konfigurationsregister verwaltet und auf eine der zwei 64KB RAM-Bereiche verteilt. Der Zugriff des VIC auf die RAM-Banks ist unabhängig davon festsetzbar.

Der Inhalt des MMU-Konfigurationsregisters ist dem PLA-Decoder bekannt und bestimmt somit die Generierung aller nötigen Chip-select-Signale für Peripherie, Prozessoranteil am Speicher, Farb-RAM, VIC-Register u.s.w.

Die Hauptaufgabe des VIC ist die für den Bildschirm nötigen Daten aus dem Speicher zu holen und dann daraus Monitor-Signale und ein moduliertes TV-Signal zu produzieren. Der 8563 Video-Controller erzeugt einen RGBI-Ausgang für einen Monitor mit 80-Zeichen-Ausgabe.

Der serielle Bus-Port, der User-Port und die Joystick-Ports entsprechen den C 64-Ports und werden über die zwei 6526 CIA-Chips betrieben.

In den folgenden Kapiteln werden wir nun die einzelnen Bausteine und ihre Funktionen genauer unter die Lupe nehmen.

2/2.2.2

Der 8502-Mikroprozessor (CPU)

Der 8502-Mikroprozessor ist eine Weiterentwicklung des 6510/6502 und ist die operierende Zentraleinheit im C 128-Modus. Da er mit den im C 64 befindlichen Prozessoren, 6510 oder auch 6502 softwaremäßig voll kompatibel ist, stellt er auch im C 64-Modus die Zentraleinheit dar. Er benutzt ebenso einen Zero-Page-Port für die Speicherverwaltung und Kommunikation mit dem Kassettenrekorder.

Zusätzlich besteht aber für den 8502 die Möglichkeit im 2-MHz-Takt zu arbeiten. Uneingeschränkt im 2-MHz-Takt arbeiten kann der 8502 allerdings nur im Zusammenhang mit dem 8563 Video-Controller (für die 80-Zeichen-Ausgabe zuständig — siehe Kapitel 2/2.2.4), da die restlichen Teile für die Verwaltung der Ein- und Ausgabe, sowie der 8564 Ersatz-Chip des älteren 6567 VIC-Chips (im C 64-Modus als Video-Interface und auch als Coprozessor fungierend) nur mit 1-MHz-Takt arbeiten können. In diesem Fall wird der VIC als Video-Interface aus dem System ausgeschaltet und behält lediglich seine Funktion als allgemeiner Taktgeber, also 2 MHz für den Prozessor und 1 MHz für die peripheren Bausteine. Im Falle eines Zugriffs auf langsamere Bausteine ist er befähigt, den 2-MHz-Takt des Prozessors auf 1 MHz zu strecken.

Zum besseren Verständnis der Funktionsweise folgt nun eine Beschreibung der Ein- und Ausgänge des Prozessors:

Pin	Name	Beschreibung
1	PHI-0	Eingang für die Systemuhr (1-2 MHz)
2	RDY	Ready; Eingang zur Steuerung des Speicherzugriffs
3	/IRQ	Interrupt request; Unterbrechungssignal
4	/NMI	Nicht maskierbare, d.h. nicht ignorierbare Unterbrechungsaufforderung
5	AEC	Adreß-Steuersignal
6	VCC	Stromversorgung
7-23 (ohne 21)	A0-A15	Die Ausgänge des Adreßbusses
21	VSS	Masse-Leitung 0 Volt
24-30	P6-P0	Ein-/Ausgänge des bidirektionalen Daten-Ports
31-38	D7-D0	Ein-/Ausgänge des Datenbusses von und zu den peripheren Bausteinen
39	R/W	Ausgang zur Kontrolle der Richtung des Datentransfers. Der Ausgang ist hoch fürs Lesen und liegt tief fürs Schreiben in und aus dem Speicher.
40	RES	Reset; Neustart

Es folgen nun ein paar Erläuterungen zu den wichtigeren Anschlüssen.

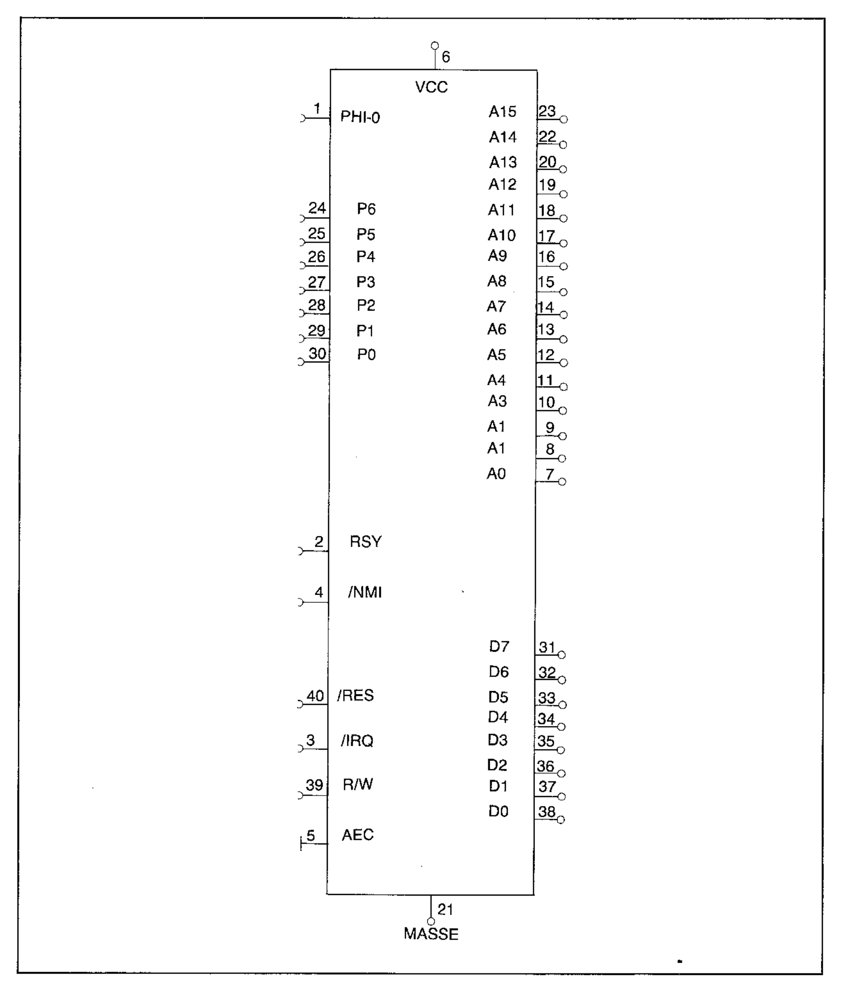


Bild 2/2.2.2-1 Der 8502-Mikroprozessor.

/IRQ

Über diesen Eingang wird der Prozessor zu einer Unterbrechung aufgefordert. Nach Abschluß der laufenden Befehlsequenz prüft der Prozessor das Interrupt-Flag seines PSR (Prozessor Status Register), und falls nicht gesetzt, beginnt er mit der Ausführung der Interrupt-Routine. Dies geschieht, indem der Prozessor, nach Sichern des Programmzählers und Status-Registers auf dem Stapel, sowie nach Setzen des I-Flags, um andere Unterbrechungen zu vermeiden, den Programmzähler aus Speicheradresse $FFFE und $FFFF lädt, und dann sein Programm an der dort befindlichen Adresse fortsetzt.

/NMI

Der Prozessor wird dadurch zu einer ähnlichen Unterbrechung veranlaßt. Im Gegensatz zu /IRQ ist die Unterbrechung nicht vermeidbar (keine Flag-Überprüfung) und der Programmzähler wird aus Adresse $FFFA und $FFFB geladen.

AEC

Der Adreßbus kann nur bei hohem AEC zur Geltung kommen, da er bei tiefem AEC in einen hochohmigen Zustand versetzt wird und somit den Speicherzugriff einer anderen Einheit (z.B. Coprozessor) freigibt.

RES

Über diesen Eingang kann der Prozessor mittels einer von der Elektronik erzeugten positiven Spannungsspitze zum Neustart veranlaßt werden. Dies bedeutet Setzen des I-Flags und Laden des Programmzählers mit dem Inhalt der Speicheradresse $FFFC und $FFFD.

2/2.2.3

Der Z80-Mikroprozessor

Es war wohl die Verbreitung und die Bekanntheit des Z80, sowie die Menge guter Software, die auf diesem Mikroprozessor lauffähig ist die Commodore veranlaßten, den Z80 in den C 128 fest einzubauen. Im Vergleich zur Möglichkeit das System mit einer Z80-Karte zu erweitern, die es ja schon bei dem C 64 gab, wird durch den festen Einbau die Handhabung des ganzen Systems wesentlich erleichtert und somit die Benutzerfreundlichkeit verbessert. Im allgemeinen ist zu bemerken, daß die Hardware des C 128 so ausgelegt wurde, um die Benutzung schon vorhandener Software zu ermöglichen, was eigentlich für Computer dieser Preiseklasse nicht unbedingt üblich ist.

Obwohl, wie man oft hören kann, alle Mikroprozessoren ähnlich konzipiert sind, benötigen, durch die Wahl der einen oder anderen technischen Lösung bedingt, die Endprodukte des einen oder anderen Herstellers eine oft recht verschiedene konkrete Umgebung um auch praktisch zu funktionieren. Diese Umgebung besteht aus Hilfsbausteinen für Ein- und Ausgabe, Speicher, Taktgeber usw. die zusammen eine sogenannte Familie bilden. Die Integration eines Bausteines aus einer anderen Produktfamilie ist meistens ohne einigen Aufwand auf der Ebene der Elektronik nicht zu bewältigen.

Im C 128 befinden sich hauptsächlich Bausteine der 85xx-Familie. Dazu kommen noch Elemente der 65xx-Vorgängerfamilie, mit der sie softwaremäßig zwar voll kompatibel sind, hardwaremäßig aber nur teilweise demselben Standard entsprechen, und schließlich der Z80 A-Prozessor als einziger völlig fremder Baustein. Es ist Commodore trotzdem gelungen, den Z80 so zu integrieren, daß er praktisch auf alle Möglichkeiten des restlichen Systems eingehen kann. Wir beschreiben nun im folgenden auf welche Art und Weise die Eingliederung des Z80 in die 85xx-Familie durchgeführt wurde.

Obwohgl der Z80 auch als Coprozessor angesprochen wird, ist dies eigentlich nicht richtig. Es gibt keine parallele Zusammenarbeit zwischen dem 8502 und dem Z80, wie es in einem echten Coprozessorsystem der Fall ist, da immer nur einer der beiden Prozessoren über den Bus verfügen kann. Die Regelung des Zugangs für den Z80 auf den Prozessorbus bedarf mehr als einem einfachen Steuersignal zur Berechtigung des Zugriffs, da das Protokoll, wonach der Z80 die Steuersignale für die Verwaltung seines Busses annimmt bzw. abgibt, nicht mit demjenigen der 85xx-Familie übereinstimmt. Der Z80 selbst ist nur an einen lokalen Bus seines Typs direkt verbunden, der einerseits die zu lesenden Daten — durch ein Latch geführt — über-

nehmen darf, andererseits die zu schreibenden Daten — während AEC hoch ist — über ein Gatter auf den Prozessorbus setzen kann.

Die nötigen Steuersignale für das Lesen und Schreiben laufen beim Z80 über die Leitungen /RD und /WR. Die Steuerung des Lesevorganges ist einfach, da der Z80 über den /RD-Ausgang seine Absicht direkt mitteilt und dadurch die Daten vom Prozessorbus auf seinen eigenen Bus holen kann. Die Steuerung des Schreibvorganges ist aber nicht mehr so problemlos. Der /WR-Ausgang wird von einer Flip-Flop- und einer Transistorschaltung unter Beachtung des AEC und des Taktimpulses verarbeitet, um die R/W-Leitung des 8502-Prozessors an die MMU zu simulieren. Außerdem ist der Z80 so konzipiert, daß er Schreib-/Lesezyklen für Speicher und Ein-/Ausgabe unterscheidet, während die 85xx-Familie die Kommunikation nur über den Speicher erledigt, wobei Ein-/Ausgabeeinheiten über die I/O-Bereiche adressiert werden, die sowohl im C 64 als auch im C 128 vertreten sind (siehe z.B. Kapitel 3/1.1 für den C 64).

Dieses Problem, sowie der Zugriff des Z80 auf die C 128 Kernal-Utilities, können aber durch Eingreifen des 8502 gehandhabt werden. Es bedarf dazu allerdings eines Mechanismuses, der die gegenseitige Übergabe der Systemkontrolle regelt. Die Umschaltung wird über die /BUSAK- und /BUSRQ-Leitungen beim Z80 bzw. AEC und RDY beim 8502 in Verbindung mit der /Z80EN-Leitung der MMU und BA-Leitung des VIC kontrolliert.

Weil der Z80 beim Einschalten empfindlich ist und keinen fremden Eingriff toleriert, wird das System durch diesen Prozessor hochgefahren. Nach einigen Initialisierungszyklen kann dann der Z80 den 8502-Prozessor im C 64- oder C 128-Modus starten oder selbst die Kontrolle behalten. Übrigens ist das Vorhandensein eines 9V-Anschlusses nur durch das Dasein des Z80 bedingt, da die notwendige Minimalspannung für den Eingang seines Taktgebers über dem von der 85xx-Familie lieferbaren Niveau liegt. Der Taktimpuls des VIC-Chips muß deswegen von einer Transistorschaltung erst auf die richtige Höhe gebracht werden bevor er zum Z80 kommt.

Wir vervollständigen nun die Beschreibung des Z80 mit der Angabe der Anschlüsse dieses wichtigen Bausteins:

Pin	Name	Beschreibung
1-5	A11-A15	
30-40	A0-A10	Der Adreßbus. Wird über 16-Bit für die Speicheradressierung und über 8-Bit für Ein-/Ausgabe (also bis zu 256 Ports) benutzt. Die unteren 7 Bits werden zum Auffrischen des Speichers benutzt.
6	PHI	Eingang für die Systemuhr.
7-14 (ohne 11)	D0-D7	Ein-/Ausgänge für den 8-Bit-Datenbus. (genaue Pin-Folge in Bild 2/2.2.3-1)
11	+5V	Stromversorgung.
16	/INT	Eingang zur Unterbrechungsaufforderung.
17	/NMI	Nicht maskierbare Unterbrechungsaufforderung.
18	/HALT	Haltezustand.

Pin	Name	Beschreibung
19	/MEMREQ	Dieser Ausgang siganlisiert, daß der Adreßbus eine gültige Adresse für einen Speicher- oder Ein-/Ausgabezyklus führt.
20	/IORQ	Ankündigungssignal für Ein-/Ausgabe.
21	/RD	Ausgang wodurch der Prozessor einen Lesevorgang ankündigt. Damit können die Daten vom Prozessorbus geholt werden.
22	/WR	Über diesen Ausgang wird signalisiert, daß der Datenbus gültige Daten führt.
23	/BUSAK	Ausgangssignal zur Anerkennung der Freigabe des Busses.
24	/WAIT	Dieser Eingang wird benutzt um den Prozessor in einen Wartezustand zu bringen.
25	/BUSRQ	Eingang, über den andere Teilnehmer Daten, Adressen und Steuersignale für die Busverwaltung vom Prozessor anfordern.
26	/RESET	Eingang, über den der Z80 zum Neustart gezwungen wird.
27	/M1	Zeigt Maschinenzyklus 1 an.
28	/RFSH	Gibt an, daß die unteren 7 Bits des Adreßbusses eine Adresse für Auffrischungslesen enthalten.
29	MASSE	Masse-Leitung 0V

Wir schließen nun das Kapitel mit einigen zusätzlichen Erklärungen, die im engen Rahmen der obigen Aufzählung nicht genügend Platz fanden.

/INT

Der Aufforderung wird nur nach Abfrage des Interrupt-Flags und falls die /BUSRQ-Leitung nicht aktiv ist, Folge geleistet. Gleichzeitig gibt der Prozessor ein Anerkennungssignal ab, und zwar über die Leitung /IORQ. Der Z80 kennt drei verschiedene Arten für die Beantwortung einer Unterbrechung, je nach Modus.

/NMI

Diese Art von Unterbrechungsaufforderung kann nicht abgelehnt werden. Sie wird immer nach Ausführung der gerade laufenden Befehlsfolge anerkannt, und zwingt den Prozessor — nach Sichern des Programmzählers auf dem Stapel — sein Programm bei Adresse $0066 fortzusetzen. Es ist noch zu bemerken, daß ein dauernder Wartezustand die Anerkennung verschieben kann, und daß /BUSRQ noch vor /NMI an den Zug kommen kann.

/HALT

Diese Zustand tritt nach Ausführung einer HALT-Anweisung ein. Während dieses Zustandes werden leere Anweisungen ausgeführt (NOP's), damit das Auffrischen des zugehörigen Speicherbereiches fortgesetzt wird. Der Zustand wird durch eine Unterbrechung oder Busabgabe beendet.

/IORQ

Zeigt das Vorhandensein einer gültigen Adresse für eine Ein-/Ausgabeoperation auf der unteren Hälfte des Adreßbusses an. Dieses Signal wird auch im Falle eines Interrupts, zusammen mit einem /M1-Signal erzeugt, um anzuzeigen, daß ein Vektor für die Beantwortung der Unterbrechung auf den Datenbus gelegt werden kann.

/M1

/M1 zeigt an, daß während des gerade laufenden Maschinenzykluses der Operationscode eines Befehls eingelesen wird. Falls es sich um einen 2-Byte-Befehl handelt, wird /M1 beim Einlesen eines jeden Bytes generiert. /M1 wird auch zusammen mit /IORQ benutzt, um die Anerkennung einer Unterbrechung zu signalisieren. Während /M1 werden keine Ein-/Ausgabeoperationen durchgeführt.

Bild 2/2.2.3-1 Der Z80-Mikroprozessor

2/2.2.4

Der 8563-Video-Controller (VDC)

Wie schon erwähnt, ermöglicht der Video-Display-Controller eine professionelle 80-Spalten- und 16-farbige Ausgabe auf einem RGBI-Monitor. Der Speicherbedarf für Zeichen und auszugebende Informationen wird von einem eigenen DRAM gedeckt, um den Speicherplatz des System-DRAM's nicht zusätzlich zu kürzen. Die Verbindung zum eigenen DRAM geht über einen lokalen Display-Bus, der dem 8502-Prozessor nicht zugänglich ist, und die Verwaltung dieses DRAM-Bereiches (Lesen, Schreiben, Auffrischen) erledigt der 8563-Video-Controller allein. Die Kapazität des angeschlossenen DRAM's kann, unter Benutzung von zwei 4416 DRAM-Einheiten, 16K oder 64K betragen, falls acht 4164 DRAM-Chips benutzt werden. Vorläufig wurde im C 128 die erste Konfiguration gewählt, und sie wird in einem internen Register festgelegt. Damit berühren wir auch das wichtigste Thema in der Beschreibung des 8563-Video-Controllers, und zwar:

Die Register des VDC

Von außen gesehen, besteht der 8563 aus nur zwei Registern, sogenannten externen Registern. Diese zwei Register werden im I/O-Bereich ab Stelle $D600 adressiert und schaffen den Zugang zu den 37 informationshaltigen internen Registern des Bausteins. Ins erste der beiden externen Register, Adreß/Status-Register genannt, kann in dessen untere 5 Bits die Adresse (d.i. in diesem Fall einfach die Nummer) des internen Registers, für das der Zugriff bestimmt ist, geschrieben werden. Die oberen Bits enthalten Statusinformationen und werden beim Lesen des Registers in einem Status-Byte ausgegeben. Das zweite externe Register, auch Datenregister genannt, enthält die Daten, die in das im Adreß/Status-Register adressierten interne Register geschrieben oder daraus gelesen werden sollen.

Je nach Funktion können die internen Register in sogenannte Satzungsregister (setup) und Ausgaberegister (display) unterteilt werden. Die ersteren enthalten Synchronisations-Daten, die den Video-Standard (PAL, NTSC usw.) festlegen. Die Register der anderen Gruppe beinhalten Informationen, welche die eigentliche Ausgabe der einzelnen Zeichen und Zeichenblöcke auf den Bildschirm bestimmen.

Eine Übersicht der Inhalte der 37 internen Register befindet sich in Tabelle 2/2.2.4-1.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R 00	Horizontal gesamt							
R 01	Horizontal ausgegeben							
R 02	Horizontale Sync.-Position							
R 03	Vertikale Sync.-Breite				Horizontale Sync.-Breite			
R 04	Vertikal gesamt							
R 05				Vertikal gesamt Einstellen				
R 06	Vertikal ausgegeben							
R 07	Vertikale Sync.-Position							
R 08							Interlace-Modus	
R 09				Zeichen gesamt vertikal				
R 10		Cursor Modus		Start-Abtastzeile für Cursor				
R 11				End-Abtastzeile für Cursor				
R 12	Ausgabe-Startadresse (HB)							
R 13	Ausgabe-Startadresse (LB)							
R 14	Cursor-Position (HB)							
R 15	Cursor-Position (LB)							
R 16	Lichtgriffel vertikal							
R 17	Lichtgriffel horizontal							
R 18	Erneuerungsort (update) (HB)							
R 19	Erneuerungsort (LB)							
R 20	Startadresse für Ausgabemerkmale (Attribute) (HB)							
R 21	Startadresse für Ausgabemerkmale (LB)							
R 22	Zeichen gesamt horizontal				Zeichen ausgegeben horizontal			
R 23				Zeichen ausgegeben vertikal				
R 24	Kopieren/ Füllen	Rev. Bildsch.	Blink-Frequ.	Vertikales weiches Scrollen				
R 25	Grafik/Text	Atrb. Enb	Halbgrafik	P1 x Dbl	Horizontales weiches Scrollen			
R 26	Vordergrundfarbe				Hintergrundfarbe			
R 27	Adressen-Inkrement-Zeile							
R 28	Zeichensatzadresse			4164/4416				
R 29				Abtastzeile für Unterstreichen				
R 30	Wortzähler							
R 31	CPU Daten Lesen/Schreiben							
R 32	Quelladresse für Block-Kopieren (HB)							
R 33	Quelladresse für Block-Kopieren (LB)							
R 34	Ausgabe ermöglichen (Display Enable): Anfang							
R 35	Ausgabe ermöglichen: Ende							
R 36					DRAM Auffrischen pro Abtastzeile			

Tabelle 2/2.2.4-1

HB = höherwertiges Byte
LB = niederwertiges Byte

Die Signale des 8563-Chips

Wie üblich, zeigen wir nun zum Abschluß des Kapitels die einzelnen Signale und Anschlüsse des betrachteten Bausteins. Da die Kommunikation des Video-Controllers in drei verschiedenen Richtungen mit drei separaten Einheiten stattfindet, und zwar mit der CPU, mit dem eigenen DRAM-Bereich über den lokalen Bus und schließlich mit dem äußeren Monitor, werden wir auch die Signale, die die entsprechenden Verbindungen realisieren, in dieser logischen Reihenfolge aufzählen.

Einige Anschlüsse des Chips sind für künftige Entwicklungen gedacht und bleiben im C 128 unbenutzt.

Pin	Name	Beschreibung
		CPU-Signale
4	CS	Chip-Select; Der 8563-Controller ist im System nur operationsfähig, falls dieser Eingang hoch ist.
7	/CS	Chip-Select; Der 8563 ist nur ansprechbar, falls dieser Eingang tief liegt.
8	/RS	Register-Select; Falls hoch, kann ins Datenregister geschrieben, oder daraus gelesen werden. Der Eingang ist auf Systemebene mit A0 verbunden.
9	R/W	Prozessorsignal; Über diesen Eingang wird die Benutzungsrichtung des Daten-Busses festgelegt.
10-18 (ohne 12)	D0-D7	Ein-/Ausgänge des Datenbusses; Erlauben den Datenaustausch zwischen dem 8563 und der CPU.
19	DSPEN	Display Enable; im C 218 nicht verwendet.
22	RES	Reset; in der aktuellen Schaltung des C 128 nicht benutzt, mit INIT verbunden.
23	INIT	Initialisierungseingang; Beim Neustart bedingt das Tiefliegen dieses Eingangs ein korrektes Initialisieren des Chips.
24	TST	Test; Nur beim Testen benutzt. Im C 128 an die Masse geleitet.
		Signale des lokalen Busses
21	DR/W	DRAM-Lese/Schreibsteuerung.
26-33	DA0-DA7	Gemultiplexter Adreßbus zum lokalen Display-DRAM.
34-42 (ohne 37)	DD0-DD7	Ein-/Ausgänge des lokalen Datenbusses.
47	RAS	Row Adress Strobe; Zeilenabtastimpuls für das lokale DRAM.
48	CAS	Column Adress Strobe; Spaltenabtastimpuls für das lokale DRAM.
		Video-Ausgänge
1	CCLK	Video-Ausgang der Zeichenuhr; Im C 128 nicht verwendet.
2	DCLK	Video-Punkt-Uhr; Bestimmt die Größe eines Pixels und ist, intern, die Zeitbasis für alle synchronisierten Signale.
3	HSYNC	Horizontales Synchronisationssignal; Über das zugehörige interne Register programmierbar.
20	VSYNC	Vertikales Synchronisationssignal; Ebenso programmierbar.
43-45	R,G,B,I	Rot, Grün, Blau, Intensität; Ausgänge zur Überlieferung eines 4-Bit-Codes (16 Möglichkeiten, d.h. Farben oder Grau-Töne) an einen externen Monitor.
12	Masse	Masse, 0V
37	VCC	Stromversorgung, +5V

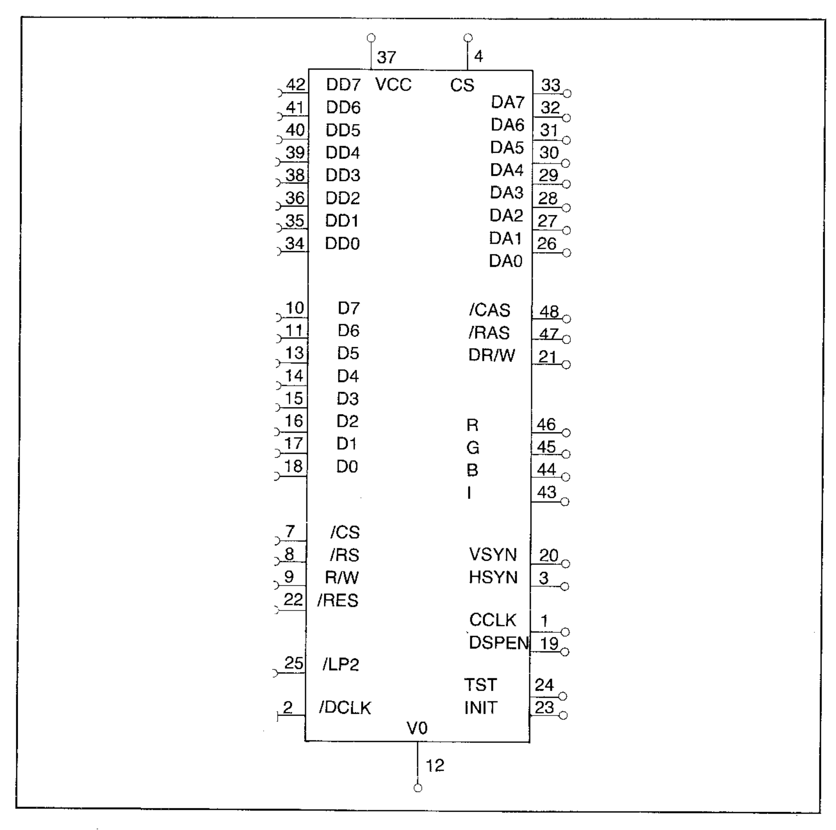


Bild 2/2.2.4-1 Der 8563-Video-Controller

2/2.2.5

Der Speichermanager (MMU)

Zusammen mit der PLA bewältigt die MMU (**M**emory **M**anagement **U**nit) die wichtigsten Verwaltungsarbeiten, die innerhalb des Systems anfallen. Mit Hilfe der MMU kann man zwischen den verschiedenen Möglichkeiten für die Speicheraufteilung, die der C 128 bietet, umschalten. Wie der Name schon andeutet, ist es die Hauptaufgabe der MMU, die komplexen Abläufe der Kommunikation anderer Bausteine mit dem Speicher zu sichern. Konkret ist die MMU für die Erzeugung der nötigen CAS-Select-Signale zur Aufteilung des RAM's, der ROMBANK-Signale zur Aufteilung des ROM's sowie für die Erzeugung des übersetzten Adreßbessus (TA8-TA15) verantwortlich. Außerdem generiert die MMU auch die nötigen Steuersignale zur Kontrolle der Prozessormodi. Die Daten der MMU werden aber nur falls AEC hoch ist in Betracht genommen, um die Überbelastung des Busses während der VIC-Zyklen zu vermeiden.

Im C 64-Modus ist die MMU zwar nicht adreßmäßig im Speicher vertreten, und somit softwaremäßig auch nicht ansprechbar, sorgt aber trotzdem für eine mit dem C 64-Standard völlig kompatible Speicherverwaltung.

Um die Funktionsweise der MMU zu verstehen und folglich auch ihren Einsatz über die Software richtig zu handhaben, ist die Kenntnis ihrer Register sowie die Bedeutung der Werte der einzelnen Bits aus diesen Registern unerläßlich. Deshalb fahren wir nun fort mit einer genauen Beschreibung der Auswirkungen, die eine Eintragung in eines dieser Register zur Folge hat.

Die Register der MMU

Die Namen der MMU-Register, sowie die von ihnen belegten Speicheradressen, können dem Bild 2/2.2.5-1 entnommen werden. Gemäß ihrer Adressen, erscheint eine Gruppe dieser Register im I/O-Bereich des Speichers ($D500 bis $D50B), die andere im Systembereich des Speichers ($FF00 bis $FF04). Nur das Konfigurationsregister, das in einem bestimmten Sinne die entscheidende Rolle spielt, ist in beiden Bereichen vertreten und kann also auch dann gelesen werden, falls der I/O-Bereich ausgeblendet ist.

Adresse	Register	Bedeutung
$D50B	VR	Versionsregister
$D50A	P1H	Zeiger für Page 1 (HB)
$D509	P1L	Zeiger für Page 1 (LB)
$D508	P0H	Zeiger für Page 0 (HB)
$D507	P0L	Zeiger für Page 0 (LB)
$D506	RCR	RAM-Konfigurationsregister
$D505	MCR	Modus-Konfigurationsregister
$D504	PCR D	Präkonfigurationsregister D
$D503	PCR C	Präkonfigurationsregister C
$D502	PCR B	Präkonfigurationsregister B
$D501	PCR A	Präkonfigurationsregister A
$D500	CR	Konfigurationsregister
$FF04	LCR D	Lade-Konfigurationsregister D
$FF03	LCR C	Lade-Konfigurationsregister C
$FF02	LCR B	Lade-Konfigurationsregister B
$FF01	LCR A	Lade-Konfigurationsregister A
$FF00	CR	Konfigurationsregister

Bild 2/2.2.5-1

Die Register der MMU
HB = höherwertiges Byte
LB = niederwertiges Byte

Das Konfigurationsregister

Der Inhalt dieses Registers bestimmt, welche ROM-, RAM- und I/O-Bereichsaufteilung des Speichers vom System gerade benutzt wird. Es befindet sich bei Adresse $D500 im I/O-Bereich und $FF00 im Systembereich. Zuweilen sind einige seiner Bit-Werte von den Leitungen MS0 und MS1 (auch als ROMBANK-Leitungen angesprochen) im C 128-Modus wiedergespiegelt, je nach Setzen von ROM und RAM. Die MS-Leitungen teilen der PLA mit, welcher Speichertyp sind in einem bestimmten Adreßbereich befindet. Im C 64-Modus allerdings sind beide Leitungen hoch, und die Aufteilung in ROM und RAM wird von der PLA nach den vom C 64 bekannten Methoden erledigt. Im folgenden werden wir nun die Bedeutung der einzelnen Bits dieses wichtigen Registers hervorheben.

Bit 0

entscheidet im C 128-Modus ob ein Zugriff auf den I/O-Bereich ($D500-$DFFF) oder auf den ROM-Bereich stattfindet. Falls dieses Bit nicht gesetzt ist (Wert 0) handelt es sich um einen I/O-Zugriff. Falls gesetzt (Wert 1), handelt es sich um einen RAM/ROM-Zugriff, dessen genauerer Ausgang von den höheren Bits desselben Registers bestimmt wird. Die MMU-Register von $D500 bis $D50B sind also nur bei nichtgesetzem Bit 0 erreichbar, während die Register $FF00-$FF04 im C 128-Modus immer im Speicher eingeblendet sind. Der Wert dieses Bits ist hardwaremäßig durch das I/O-Signal (Pin 13) angegeben und wird im C 64-Modus von der PLA ignoriert, die den I/O-Bereich mittels HIROM und CHAREN festlegt. Im Falle eines Resets nimmt dieses Bit den Wert 0 an.

Bit 1

kontrolliert im C 128-Modus den Zugriff des Prozessors auf den unteren ROM-Bereich ($4000-$7FFF). Falls nicht gesetzt, ist das System-ROM in diesem Bereich eingeschaltet. Dieses Bit beeinflußt die beiden Status-Leitungen MS0 und MS1, die von der PLA decodiert werden, um anschließend die richtigen ROM-Chip-Selects zu erzeugen. Bei gesetztem Bit 1 wird der genannte Bereich als RAM interpretiert, und das CAS-Signal wird an das von weiteren Bits des Registers festgelegte RAM-Bank zugeleitet. Im C 64-Modus wird der Wert von Bit 1 nicht berücksichtigt. Der Reset-Wert ist 1, und blendet in diesen Bereich das untere C 128 Basic-ROM ein.

Bits 2 und 3

bestimmen im C 128-Modus den Speichertyp des mittleren Bereiches ($8000 bis $BFFF). Hardwaremäßig wird Bit 2 von MS1 und Bit 3 von MS0 angezeigt. Beide gesetzt, bedeutet RAM in diesem Bereich mit den entsprechenden CAS-Signalen, keines gesetzt dagegen System-ROM. Falls nur Bit 2 den Wert 1 hat, werden in diesem Speicherabschnitt die internen ROM-Funktionen erreicht, und falls nur Bit 3 gesetzt ist, die externen. Im C 64-Modus werden beide Bits ignoriert, und beim Neustart nehmen sie den Wert 0 an, um hier das obere Basic-ROM einzublenden.

Bits 4 und 5

haben folgerichtig denselben Einfluß auf den oberen Speicherabschnitt ($C000 bis $FFFF), nur werden die hier liegenden ROM-Abschnitte den Zeichensatz und den Kernal erfassen. Es ist noch zu bemerken, daß im Falle einer Überlagerung mit der Angabe des Bits 0, entscheidend für Bereich $D500-$FFFF, das I/O-Bit den Vorrang hat, und daß es im oberen Block einen unantastbaren Abschnitt zwischen $FF00 und $FF04 gibt.

Bit 6

entscheidet schließlich, welches der RAM-Banks ausgewählt wird. Falls nicht gesetzt, ist Bank 0 gewählt, und das /CAS0-Signal wird auf tief gesetzt. Falls gesetzt,

ist es Bank 1 und /CAS1 fällt auf tief. Bit 7 hat vorläufig keine Funktion und bleibt für künftige Erweiterungen frei.

Ändern der Speicheraufteilung

Da wir jetzt wissen, welche Bedeutung dem Konfigurationsregister bei der Speicheraufteilung zugeschrieben ist, werden wir nun zeigen, wie der Benutzer sich Zugang zu diesem Register schaffen kann, um auf diese Weise die gewünschte Speicherkonfiguration zu bestimmen.

Mit Hilfe von zwei Registergruppen, und zwar vier Präkonfigurationsregister und vier Lade-Konfigurationsregister, ist ein von Seiten des Benutzers festsetzbares Laden des Konfigurationsregisters bei geringem Zeitverlauf und Speicheraufwand gewährleistet. Die PCR's A bis D speichern die verschiedenen Möglichkeiten der Speicheraufteilung, die der Benutzer in Betracht zieht und dann, zum gegebenen Zeitpunkt mit einem einzigen Befehl einsetzen wird. Das Format eines jeden Präkonfigurationsregisters ist folglich auch dasselbe wie das Format des Konfigurationsregisters. Die PCR's befinden sich, wie aus Bild 2/2.2.5-1 ersichtlich, im I/O-Bereich ab Abdresse $D501 bis $D504. Das Schreiben in eines dieser Register hat vorerst keine direkte Auswirkung. Es wird nur eine Konfiguration gespeichert, die dann durch einen Befehl auch realisiert werden kann. Bei einem Neustart werden alle vier Register auf 0 gesetzt.

Die Lade-Konfigurationsregister entsprechen paarweise den Präkonfigurationsregistern und befinden sich im Systembereich ab $FF01 bis $FF04, sind somit, zusammen mit dem Konfigurationsregister bei $FF00 immer zugänglich, unabhängig von der jeweiligen Speicheraufteilung. Beim Lesen eines LCR's wird der Wert des entsprechenden PCR's ausgegeben, und beim Schreiben in ein solches Register wird der Wert des zugehörigen PCR's in das Konfigurationsregister gebracht, und dadurch auf bekannter Weise eine bestimmte Speicheraufteilung verwirklicht. Es ist also möglich, auch aus einer Konfiguration in der der I/O-Bereich nicht erscheint und die PCR's nicht zugänglich sind, umzuschalten, da die Lade-Konfigurationsregister immer zugänglich bleiben. Im C 64-Modus haben diese beiden Registergruppen keinen Einfluß. Bei einem Reset werden sie mit dem Wert 0 initialisiert.

Das Modus-Konfigurationsregister

Dieses Register befindet sich im I/O-Bereich bei der Adresse $D505. Wie der Name es schon verrät, bestimmt der Wert dieses Registers den Arbeitsmodus des Computers, d.h. welcher Prozessor die Oberhand hat und mit Hilfe welcher Betriebssystem-Variante er gerade arbeitet.

Bevor wir zur Beschreibung der Funktionen der einzelnen Bits übergehen (siehe Bild 2/2.2.5-2), ist zu bemerken, daß einige davon, namentlich FSDIR, GAME, EXROM und 40/80, wie bidirektionale Ports arbeiten. Ein in den Port geschriebener Wert wird von der Hardware-Leitung angegeben, und kann davon auch wieder gelesen werden. Nur falls eine externe Erweiterung eine Port-Leitung auf tief bringt, wird

das Lesen dieser Leitung den 0-Wert anzeigen. Nach Entfernen der externen Erweiterung wiedergibt sie den vorher gespeicherten Wert. Bei der Beschreibung der einzelnen Bits werden wir genauer auf die Ein-/Ausgangs-Funktionen der Port-Bits eingehen.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
40/80 Sensor	Betriebs-System	/EXROM Sensor	/GAME Sensor	FSDIR	Frei für Erweiterungen		Prozessor-Modus

Bild 2/2.2.5-2 Das Modus-Konfigurationsregister

Bit 0

setzt fest, welcher Prozessor das System kontrolliert, und zwar über die Ausgangsleitung /Z80EN. Hat das Bit den Wert 0, liegt die /Z80EN-Leitung tief, und der Z80-Prozessor ist aktiv. Im Z80-Modus wird jede Adresse aus RAM-Bank 0, von $0000 bis $0FFF, in die entsprechende Adresse von $D000 bis $DFFF übersetzt, wo sich das Z80 CP/M BIOS im System-ROM befindet. Die Leitungen MS0 und MS1 werden System-ROM anzeigen (also tief), und die Page 0 und Page 1 Offset-Zeiger werden ausgeschaltet. Das RAM kann trotzdem noch vom A16-Bit (Bit 6) des Konfigurationsregisters, über CAS0 und CAS1, in Banks geteilt werden. Falls Bank 1 gewählt wurde, ist das BIOS ROM ausgeblendet und das System kann von $0000 bis $FFFF RAM adressieren. Die Page 0- und Page 1-Zeiger sind in diesem Fall wieder wirksam. Beim Anschalten sowie im Falle eines Resets hat das Bit den Wert 0, da, wie schon in Kapitel 2/2.2.3 erklärt wurde, das System durch den Z80-Prozessor hochgefahren werden muß, der dann nach Abschluß seiner Initialisierungsroutinen die Kontrolle abgeben kann.

Bits 1 und 2

sind nicht benutzt und bleiben als zukünftige mögliche Port-Leitungen frei. Beim Lesen liefern sie unveränderlich den Wert 1.

Bit 3

trägt, zusammen mit der entsprechenden Hardware-Leitung, den Namen FSDIR (Fast Serial Direction). Als Ausgangs-Leitung des Ports kontrolliert das Bit die Richtung der Datenübertragung im Puffer der schnellen seriellen Schnitt-Stelle zum Diskettenlaufwerk. Als Eingang fungiert es als Sensor für ein Steuersignal der obengenannten Schnittstelle.

Bits 4 und 5

sind Sensor-Bits für /GAME und /EXROM. Als Eingänge wiedergeben sie die Hardware-Leitungen zur Kontrolle von /GAME und /EXROM nach den C 64-Regeln. Da C 128-Erweiterungen /GAME und /EXROM nicht benutzen, wird das

Aufspüren dieser Leitungen im C 128-Modus mit der Umschaltung auf den C 64-Modus beantwortet.

Bit 6

bestimmt den Betriebssystem-Modus und ist von dem C 128-Signal (Pin 47) logisch invertiert wiedergegeben. Wenn das Bit gesetzt ist, verschwindet die MMU aus dem adressierbaren Speicher und das System schaltet auf C 64-Modus um. Beim Neustart erhält dieses Bit den Wert 0, so daß alle MMU-Register ansprechbar sind.

Bit 7

ist das Ergebnis, das der Abtastsensor des 40/80-Spalten-Schalters liefert. Der Wert ist 1 bei offenem Schalter und 0 bei geschlossenem. Die Ausgangsfunktion dieses Bits ist vorläufig noch nicht festgesetzt.

Das RAM-Konfigurationsregister

Das RAM-Konfigurationsregister befindet sich im I/O-Bereich bei Adresse $D506 und beinhaltet die Parameter der RAM-Aufteilung, sowohl für den Prozessor wie als auch für den Block-Zeiger des VIC-Chips. Wir nehmen, wie üblich, die Funktionen der einzelnen Bits unter die Lupe.

Bits 0 und 1

Der Wert von Bits 0 und 1 bestimmt zusammen den RAM-Anteil, der für die RAM-Banks gemeinsam ist, falls überhaupt gemeinsames RAM von den nächsthöheren Bits erlaubt ist. Die Kombination 00 legt 1K gemeinsames RAM fest. Die Kombinationen 01, 10, 11, legen gemeinsame Bereiche von 4K, 8K und 16K entsprechend fest. Die RAM-bestimmenden Bits des Konfigurationsregisters werden von Bits 0 und 1 des RAM-Konfigurationsregisters überstimmt, in dem Sinne, daß der gemeinsame RAM-Anteil sich immer in Bank 0 befindet. Im C 64-Modus haben diese Bits keinen Einfluß. Ihr Resetwert ist 0.

Bits 2 und 3

entscheiden, welcher RAM-Anteil gemeinsam sein wird, falls überhaupt. Wenn beide Bits Wert 0 haben, gibt es keinen gemeinsamen Abschnitt. Ist Bit 2 gesetzt, so liegt der gemeinsame Abschnitt im unteren RAM-Bereich, für gesetztes Bit 3, entsprechend oben, und falls beide gesetzt, oben und unten. Der Zugriff auf den gemeinsamen Speicher ist durch Tiefsetzen von CAS0 und Hochbringen von CAS1 realisiert.

Bits 4 und 5

Die Funktionen dieser Bits sind noch nicht festgelegt. Beim Lesen liefern sie unveränderlich den Wert 0.

Bit 6

funktioniert als RAM-Bank-Zeiger für den VIC-Chip. Damit kann entweder RAM-Bank 0 oder RAM-Bank 1 für den Zugriff des VIC gewählt werden, und zwar unabhängig von der Bank-Aufteilung, die für den Prozessor gilt. Im 80-Zeichen-Modus schaltet sich der VIC-Chip durch konstantes Hochschalten von AEC selbst aus, und übergibt die Ausführung der Ausgabefunktionen an den 8563-VDC, der ebenfalls unabhängig seine Bank-Aufteilung des RAM bestimmen kann.

Bit 7 ist funktionslos.

Das Versionsregister

Dieses Register ist ein Statusregister, das nur gelesen werden kann. Es befindet sich bei Adresse $D50B im I/O-Bereich. Es hat keine funktionelle Aufgabe, sondern enthält im unteren Halbbyte einen Code, der die Version der eingebauten MMU-Ausführung identifiziert, und im oberen Halbbyte einen Code, der die Anzahl der Speicherblöcke anzeigt, beim jetzigen Stand, Wert 2 (zwei 64K Blöcke). Das Dasein dieses Registers wird es ermöglichen, daß die Software, die für künftige C 256- oder andere Erweiterungen geschrieben werden wird, auch für den jetzigen C 128 abwärts kompabitel bleiben wird.

Die Page-Zeiger

Diese vier Register befinden sich im I/O-Bereich von $D507 bis $D50A, und erlauben das Verlegen von Page 0 und Page 1 in jedem Prozessor-Modus. Dies ist besonders für den 8502-Prozessor interessant, da dadurch einige Nachteile der 65xx-Familie, wie z.B. begrenzte Stapelgröße, und im allgemeinen ziemlich ausgebuchte Page 0, beseitigt werden können.

Die Verlegung der Page 0 wird mit Hilfe der beiden Register P0L und P0H durchgeführt (siehe Bild 2/2.2.5-1). Bits 0 bis 7 des P0L entsprechen den übersetzten Adressen TA8 bis TA15 zur Bestimmung des Zugriffs auf Page 0. Falls in dieses Register geschrieben wird, gilt die versetzte Page 0 ab dem nächsten Tief der Systemuhr. Im Register P0H ist nur Bit 0 ansprechbar und entspricht TA16. Eine Eintragung in dieses Register tritt nur nach einem Schreiben ins P0L-Register in Kraft, und kontrolliert die Erzeugung von CAS0, bei Wert 0, oder von CAS1, für Wert 1 dieses Bits.

Die Verlegung der Page 1 wird mit Hilfe der Register P1L und P1H entsprechend verwirklicht. Nach einem Reset werden alle vier Register auf die wahren Page 0 und Page 1 initialisiert.

Wir haben zwar einige der Verbindungsleitungen der MMU zu den anderen Bausteinen erwähnt, für ein gründliches Verständnis der Rolle dieses wichtigen Bausteins innerhalb des C 128 werden wir nun die einzelnen Leitungen systematisch aufzählen und die wichtigeren davon eingehender erläutern.

Pin	Name	Beschreibung
1	VCC	Stromversorgung, +5V
2	RESET	Eingang zur Initialisierung der MMU-Register bei Neustart. Die Initialisierungswerte für die einzelnen Register wurden in den vorangehenden Abschnitten angegeben.
3-10	TA8-TA15	Ausgänge des übersetzten Adreßbusses
11	/CAS0	Signal zur Kontrolle der RAM-Aufteilung in Banks. /CAS0 bedingt den Zugriff auf den ersten 64K-Bereich.
12	/CAS1	Wie /CAS0, aber für den zweiten 64K-Bereich.
13	I/O	Auch MS2 genannt, wiedergibt den Wert von Bit 0 des Konfigurationsregisters, wie dort schon erklärt.
14-15	MS1-MS0	Auch ROMBANKHI und RAMBANKLO genannt.
16	AEC	Signal zur Regelung der Bus-Priorität.
17	MUX	Das gemultiplexte Speichersignal. Es wird als Taktgeber für verschiedene Abschnitte der MMU benutzt.
18-21	A0-A3	
24-31	A8-A15	Adreßleitungen vom Prozessor. Sie bestimmen die Erzeugung von Chip-Selects und gemultiplexten Adreßleitungen.
22-23	A4/5-A6/7	Kombinierte Adreßleitungen vom Prozessor. So zusammengelegt, um die Anzahl der Pins zu reduzieren.
32	R/W	Eingang für Lese-/Schreib-Signal des Prozessors. Liegt hoch für eine Lese- und tief für eine Schreib-Operation des Prozessors.
33	PHI-0	Eingang der Systemuhr. Die Prozessor-Adresse gilt während der steigenden Phase, die Daten während der sinkenden.
34	MASSE	Masse, 0V
35-42	D0-D7	Über diese Eingänge werden Daten vom Prozessor in die internen Register der MMU geschrieben.
43	/Z80EN	Falls dieser Ausgang auf tief geht, wird der Z80-Modus eingeleitet. Sonst ist der 8502 der Hauptprozessor.
44	FSDIR	Port-Leitung von Bit 3 des Konfigurationsregisters.
45	/GAME	Port-Leitung von Bit 4 des CR. Da im C 128-Modus frei, wird es in diesem Modus als Steuersignal für den Farb-RAM-Bereich benutzt.
46	/EXROM	Port-Leitung des Bit 5 des CR.
47	C 128	Auch MS3 genannt. Dieser Ausgang ist im C 128-Modus hoch. Falls die MMU ihn auf tief setzt, schaltet das System auf C 64-Modus um.
48	40/80	An Bit 7 des Konfigurationsregisters gekoppelt.

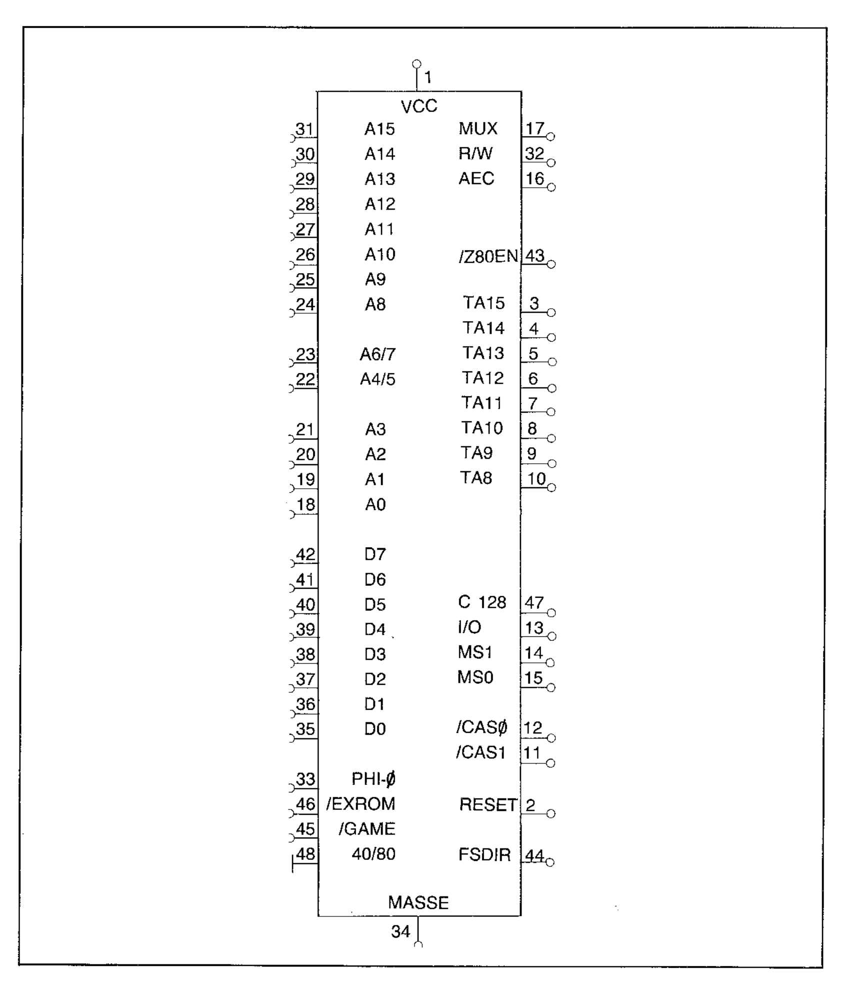


Bild 2/2.2.5-3 Der Speichermanager (MMU)

TA8-TA15

Diese Leitungen liefern die übersetzten Adressen des gemultiplexten Adreßbusses und des geteilten Busses. TA8 bis TA11 und TA12 bis TA15 haben intern einen verschiedenen Status, um den Ablauf der VIC-Zyklen in Abhängigkeit von AEC zu erlauben.

MS0-MS1

Über diese Ausgänge kontrolliert die MMU die Aufteilung des Speichers in Bereiche. Diese Signale leiten jeden ROM-Zugriff im C 128-Modus auf einen bestimmten ROM-Bereich. Falls beide tief liegen, wird der Zugriff aufs System-ROM geleitet. Wenn MS0 hoch ist, dann wird auf externe ROM-Funktionen zugegriffen, und falls nur MS1 hoch ist, auf interne ROM-Funktionen. Sind beide hoch, so wird auf das RAM des entsprechenden Abschnitts zugegriffen.

AEC

Adress Enable Control = Steuersignal zur Freigabe der Adressierung. Dieses Signal zeigt an, ob der 8502-Prozessor oder der VIC auf den geteilten Bus Zugang hat. Falls tief, kann der VIC oder ein Ausgabe-Chip auf den Speicher zugreifen, und es findet keine Übersetzung für Zeiger oder BIOS-Adressen statt.

2/2.2.6

PLA-Programable Logic Array

PLA heißt Programable Logic Array, also programmiertes Logikfeld. Dieses Logikfeld besteht aus verschiedenen Typen von Gattern, die durch die Programmierung in bestimmter Weise verknüpft werden. Ein PLA ersetzt in der Regel 10 - 30 herkömmliche Logik-ICs und ist außerdem wesentlich flexibler.

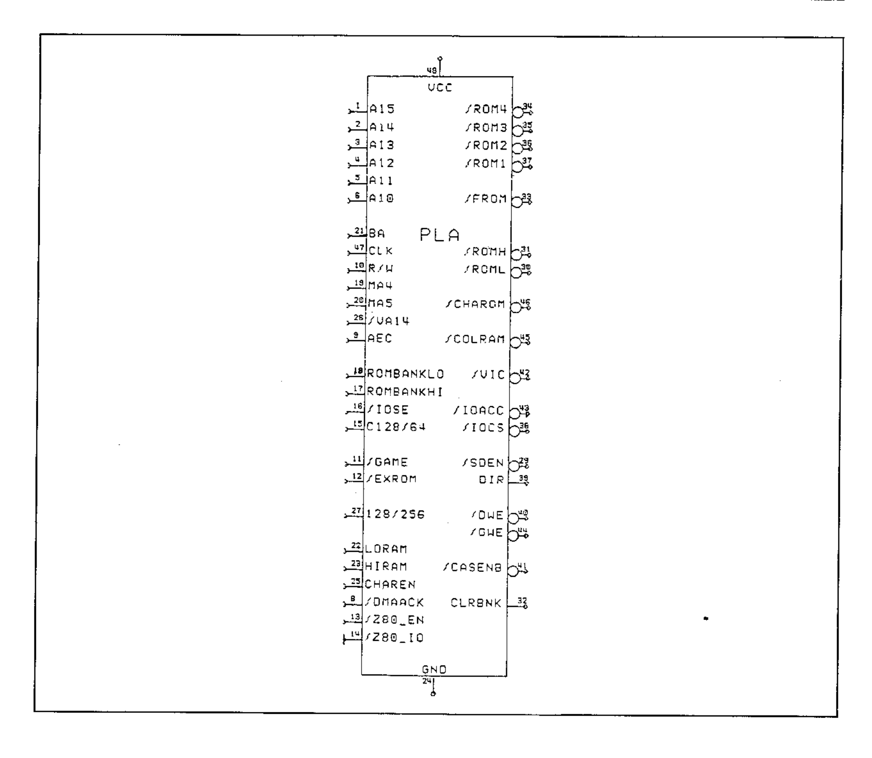
Das PLA im Commodore 128 PC

Das PLA im C 128 PC übernimmt wesentliche Aufgaben bei der Adreß- und Signalsteuerung. Es wird von den beiden Prozessoren sowie von der Memory Management Unit (MMU, siehe Kapitel 2/2.3.5) gesteuert.

Während im C 64-Modus das PLA das komplette Speichermanagement steuern muß, wird diese Aufgabe im C 128-Modus voll von der MMU übernommen. So ist das PLA im C 128-Modus frei für die Feinarbeiten im Speicher. Damit die Funktionen dieses wichtigen Bausteins klarer werden, sollen hier die einzelnen Ein- und Ausgänge näher beschrieben werden:

Pin	Name	Beschreibung
1-6	A15-A10	Adreßleitungen A15-A10 vom 8502 bzw. Z-80
7	VICFIX	Timing-Steuerung für VIC-Chip (40 Zeichen)
8	/DMAACK	Aknowledge Direktspeicherzugriff
9	AEC	Adreß-Steuersignal des VIC-Chip
10	R/W	Schreib/Lese-Leitung vom 8502 bzw. Z-80
11	/GAME	Eingang für die Erkennung externer ROM's
12	/EXROM	Eingang für die Erkennung externer ROM's
13	/Z-80 EN	Gibt Bus für Z-80 frei
14	/Z-80 I/O	I/O-Zugriff des Z-80
15	C128/64	Steuersignal der MMU: 128/64-Modus
16	/I/O SEL	Steuersignal der MMU: I/O-Bereich einblenden
17	ROMBANKHI	Steuersignal der MMU: Speicherselektion Bit 1
18	ROMBANKLO	Steuersignal der MMU: Speicherselektion Bit 0
19-20	VMA4-VMA5	Adreßleitungen A4-A5 vom VIC-Chip
21	BA	VIC gibt Bus frei
22	LORAM	Konfigurationssignal vom Prozessorport, Bit 0
23	HIRAM	Konfigurationssignal vom Prozessorport, Bit 1
24	GND	Masse-Leitung 0 Volt (Ground)
25	CHAREN	Konfigurationssignal vom Prozessorport, Bit 2
26	/VA14	VIC-Adreßleitung A14 von der CIA
27	128/256	Steuersignal für ROM mit 128 oder 256 KBit
28	-----	nicht angeschlossen
29	/SD EN	Datenpuffer-Steuersignal

Pin	**Name**	**Beschreibung**
30	/ROML	Selektionssignal für externe ROM's Low
31	/ROMH	Selektionssignal für externe ROM's High
32	CLRBNK	Selektiert zwischen den zwei Color-RAM-Bereichen
33	/FROM	Selektionssignal für das interne Funktions-ROM
34-37	/ROM4-/ROM1	Selektionssignale für die 4 internen ROM's
38	/IOCS	Selektionssignal für I/O im Bereich $D000-$DFFF
39	/DIR	Richtingssignal für Datenpuffer
40	/DWE	Schreibsignal für die dynamischen RAM's
41	/CASENB	Selektiert zwischen den beiden 64-K-Bereichen
42	/VIC	Selektionssignal für den VIC
43	/I/OACC	I/O-Zugriff, veranlaßt VIC zu 1 MHz-Takt
44	/GWE	Schreibsignal für Color-RAM
45	/COLRAM	Selektionssignal für Color-RAM
46	/CHAROM	Selektionssignal für Character-ROM
47	/CAS	Spaltenselektion für die RAM-Chips
48	+5V	Stromungsorgung (+5 Volt)



Nach dieser Übersicht über alle Pins des PLA's werden wir jetzt einige Leitungen etwas genauer unter die Lupe nehmen:

/EXROM und /GAME (Pins 11 und 12)

Diese beiden Eingänge sind im C-64-Modus dafür verantwortlich, extern, d.h. an Expansionsport angeschlossene ROM's bzw. EPROM's zu erkennen. Liegt mindestens eines der beiden Signale auf low, so wird in bestimmten Speicherbereichen statt des internen RAM's bzw. ROM's ein externer Speicher selektiert.

Näheres hierzu ist den Speicherbelegungsplänen zu entnehmen. Beim C 128-Modus haben diese Eingänge keine Funktion, denn wenn beim Einschalten des C 128 eines dieser beiden Signale auf low liegt, was über die MMU abgefragt werden kann, wird sofort in den C 64-Modus geschaltet.

/ROML und /ROMH (Pins 30 und 31)

Diese beiden Ausgänge sind die korrespondierenden Selektionssignale zu /EXROM und /GAME. Mit diesen Signalen werden im C 64-Modus die externen ROM's selektiert. Auch im C 128 können über diese Ausgänge externe ROM's angesprochen werden, und zwar im Bereich $8000-$BFFF für ROML und $C000 $FFFF für ROMH (siehe auch /FROM). Um ein externes EPROM anzusprechen, muß die MMU entsprechend programmiert werden.

LORAM, HIRAM und CHAREN (Pins 22, 23 und 25)

Diese drei Eingänge werden direkt vom prozessoreigenen Port des 8502 gesteuert (Bit 0-2). Im C 64-Modus wird mit diesen Signalen das komplette Speichermanagement gesteuert. Im C 128-Modus übernimmt die MMU das Speichermanagement, so daß die drei Signale des Prozessors für andere Aufgaben benutzt werden können.

LORAM und HIRAM steuern die Color-RAM-Bank. Im Gegensatz zum C 64 ist im C 128 ein 2-KB-RAM-Bereich für das Speichern der Farbe eingebaut. Mittels LORAM und HIRAM kann programmgesteuert zwischen den zwei 1-KB-Bereichen umgeschaltet werden. Mit dem Pegel von LORAM wählt man den Bereich, auf den der Prozessor zugreift, mit dem Pegel von HIRAM den Bereich, aus dem der VIC seine Farbinformationen bezieht. Dies hat den Vorteil, daß das Farb-RAM geändert werden kann, ohne daß die Änderung sofort auf dem Bildschirm zu sehen ist. So wird z.B. flackerfreies Scrollen des Bildschirminhalts in alle Richtungen möglich.

CHAREN bestimmt, ob das Character-ROM in den jeweiligen 16-KByte-Bereich des VIC eingeblendet wird. Ist diese Leitung geschaltet, so stellt sich der entsprechende Speicherbereich für die VIC und CPU gleich dar. Ist CHAREN low, wird relativ zur Position des VIC im Speicher bei $1000-$1FFF das Charakter-ROM eingeblendet. Während im C 64-Modus das Character-ROM im ersten und im dritten 16-Kbyte-Bereich fest eingeblendet ist, kann man im C 128-Modus frei wählen, ob eine Einblendung erfolgen soll oder nicht.

C128/64, I/O SEL, ROMBANKLO und ROMBANKHI (Pins 15 bis 18)

Diese vier Eingänge des PLA korrespondieren direkt mit Registern in der MMU und werden deshalb in Kapitel 2/2.2.5 beschrieben.

/CASENB (Pin 41)

Dieser Ausgang wählt zwischen den beiden 64-KByte-Bereichen. Um zu verstehen, wie dies funktioniert, werden wir etwas näher auf die Ansteuerung des dynamischen RAM's (128 KB) eingehen.

Die Adresse, an der der Speicherzugriff erfolgen soll, wird den RAM-Chips in zwei Teilen mitgeteilt, nämlich Reihen- und Spaltenadresse. Nachdem der RAM-Chip die Reihenadresse erhalten hat (RAS-Signal), wird intern die komplette 'RAM-Reihe' aufgefrischt (refreshed). Dies ist bei dynamischen RAM's notwendig, da sie sonst ihren Inhalt verlieren. Die nachfolgende Spaltenadresse erhält aber nur die 64-KByte-Bank, auf die zugegriffen wird.

Erst nachdem das entsprechend CAS-Signal anliegt, geben die RAM-Chips ihre Daten aus bzw. lassen sich beschreiben. Daraus ergibt sich: Refresh bei beiden Banks, Zugriff bei einer Bank. Ähnlich arbeitet übrigens auch der VIC, der eigentlich für den Refresh der RAM‘s zuständig ist. Diese Technik nennt man „Ras-only-Refresh".

/ROM1 bis /ROM4, 128/256 (Pins 34 bis 37 und Pin 27)

Die Anschlüsse /ROM1 bis /ROM4 dienen den Selektionssignalen für die vier internen 16 KByte-ROM's (C 64-Kernal und BASIC, C 128 LO, C 128 MID und C 128 HI). Der Eingang 128/256 ist für den Fall vorgesehen, daß statt 16 KByte-ROM's höher integrierte 32-KByte-ROM's benutzt werden sollen. In diesem Fall würden dann nur noch /ROM1 und /ROM3 mit jeweils 32 KByte angesprochen.

/FROM (Pin 33)

Das Signal /FROM selektiert ein internes Funktions-ROM im Bereich $8000-$FFFF. Ob eine Selektion erfolgt, wird von der MMU gesteuert. Dieses ROM kann eine Kapazität von bis zu 32 KByte haben (z.B. 27256), es ist allerdings zu beachten, daß bestimmte Speicherbereiche zeitweise ($D000-$DFFF für I/O-Bereich) oder grundsätzlich ($FF00-$FF04 für MMU) ausgeblendet werden.

2/3
Peripheriegeräte



Der C 64 und seine Peripherie.

Nachdem der äußere und der innere Aufbau des C 64 und des C 128 PC beschrieben wurden, wollen wir uns nun den Peripheriegeräten zuwenden. Sinn dieses Kapitels ist es, Ihnen zunächst die grundlegenden Informationen zu den einzelnen Peripheriegeräten zu liefern, um Ihnen eine eventuelle Kaufentscheidung zu vereinfachen, indem Sie die Daten der verschiedenen Geräte direkt miteinander vergleichen können. Zuerst soll hierbei auf Commodoreeigene Geräte eingegangen werden, aber auch Peripheriegeräte anderer Hersteller werden vorgestellt.

Im weiteren Verlauf der Kapitel über die einzelnen Geräte werden dann Besonderheiten und einige Tips und Kniffe besprochen. Außerdem finden Sie unter den in Kapitel 4/6 dargestellten Utilities und den Musterprogrammen aus Teil 5 sowie den Hard- und Softwareergänzungen aus Teil 7 weitere Hinweise zur Handhabung, Ergänzung und Benutzung der Peripheriegeräte.



Der C 128 PC und seine Peripherie.

2/3.1

Floppy-Laufwerke

Im Rahmen der Besprechung von Peripheriegeräten wollen wir als erstes die Floppy-Laufwerke behandeln, da sie im allgemeinen auch das erste Peripheriegerät darstellen, was zusätzlich zum Rechner angeschafft wird.

2/3.1.1

Das Laufwerk 1541

Das Floppy-Laufwerk mit der Nummer 1541 von Commodore ist die Standard-Diskettenstation, die im Zusammenhang mit dem C 64 eingesetzt wird. Ein früheres Modell ist das Laufwerk 1540, das aber nicht mehr im Handel anzutreffen ist. Das Modell 1541 stellt die verbesserte — fehlerbereinigte — neue Version dar.

2/3.1.1.1

Allgemeine Daten

Die 1541 ist ein sogenanntes Einzellaufwerk, das mit $5^1/_4$“-Disketten arbeitet. Zur Zeit haben $5^1/_4$“-Disketten die weiteste Marktverbreitung, obwohl Geräte mit kleineren Disketten (3“ oder $3^1/_2$“) mittlerweile immer größeren Anklang finden.

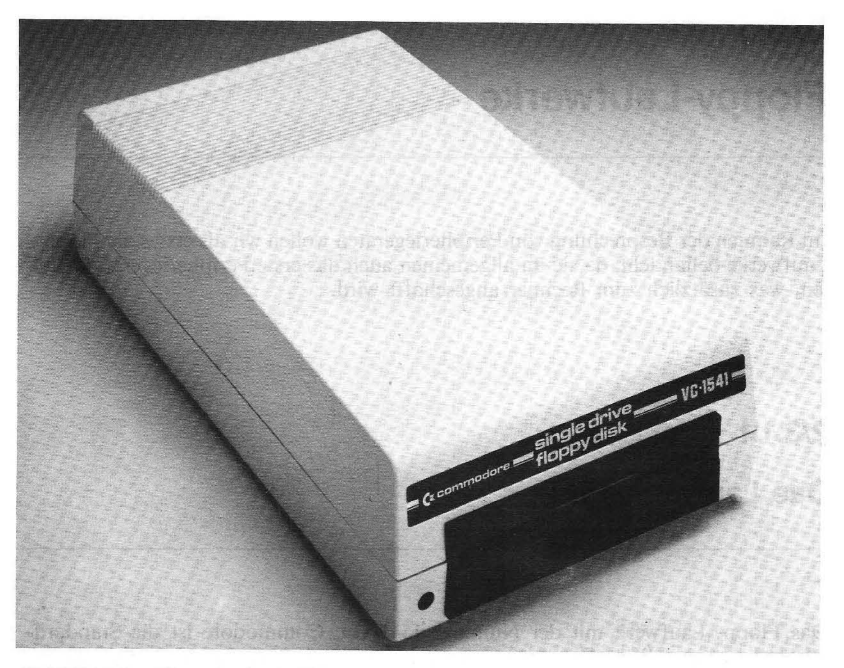


Bild 2/3.1.1 Das Floppy-Laufwerk 1541

Die Speicherkapazität von 170 KB ist langjähriger Standard, der auch heute noch bei den meisten Home-Computern anzutreffen ist.

Wichtig ist, daß das Laufwerk nicht vom C 64 gesteuert wird, sondern ein eigenes Betriebssystem aufweist, was ein Disketten-Laufwerk in der Regel schneller macht. Leider kann man trotzdem die 1541 nicht als schnell bezeichnen. Ein weiterer Nachteil der 1541 ist die mangelnde Wärmeabfuhr, so daß bei täglichem, vielstündigem Einsatz Lesefehler auftreten können. Hier schafft ein kleiner Ventilator Abhilfe, aber Betriebszeiten von mehr als acht Stunden sind eine Seltenheit bei normaler Heimanwendung.

Im folgenden eine Übersicht über die wichtigsten Daten der 1541:

Technische Daten	
DOS	
(Disk Operating System)	Betriebssystem mit 16 KByte
Version	DOS 2.6
RAM-Puffer	2 KByte
Formatierung	Laufwerk formatiert jede softsektorierte Standarddiskette 5,25" mit 35 konzentrischen Spuren, die zwischen 17 (innere Spur) und 21 (äußere Spur) Sektoren aufweisen
Inhaltsverzeichnis	auf Spur 18
Schnittstelle	serieller IEEE-Bus (IEC)
Gerätenummer	8 (kann hard- oder softwaremäßig geändert werden).
Speicherkapazität	
Frei verfügbar	168 656 Bytes (sequentielle Files) 167 132 Bytes (relative Files)
Befehlsvorrat	
Files	OPEN — Datei öffnen CLOSE — Datei schließen LOAD — Programm laden SAVE — Programm sichern LOAD „$" — Inhaltsverzeichnis laden NEW — Diskinhalt löschen VERIFY — Programm im RAM mit File vergleichen RENAME — Datei umbenennen COPY — Datei kopieren SCRATCH — Datei löschen
Direktzugriff	BLOCK-READ — Sektor lesen BLOCK-WRITE — Sektor schreiben BLOCK-EXECUTE — Programm in einem Sektor ausführen BLOCK-ALLOCATE — Sektor sicherstellen BLOCK-FREE — Sektor freigeben MEMORY READ — Sektorbereich auslesen MEMORY WRITE — Sektorbereich schreiben MEMORY-EXECUTE — Sektorbereich (Programm) ausführen BUFFER-POINTER — Zeiger in Puffer setzen USER BLOCK bezieht sich dabei direkt auf die Information auf die Diskette (Block—Sektor), MEMORY auf den Speicher des DOS
Benutzerhinweis	Offene Dateien und Funktionsstörungen sowie der Betriebszustand der Floppy werden durch LED's angezeigt. Fehler- und Statusinformationen können über einen eigenen Datenkanal erfragt werden.
Filetypen	sequentiell, relativ (Direktzugriff/Random Access), Programmdateien, User-Dateien
Netzspannung	220 V, 50 Hz
Abmessungen	
Breite	200 mm
Tiefe	374 mm
Höhe	97 mm

Mitgeliefertes Zubehör
Ausführliches Bedienungshandbuch, Testdiskette und Verbindungskabel zur Zentraleinheit.
Die Floppy VC-1541 ist anschließbar an den VC 20 und den C 64 sowie den C 128 PC.

Das Floppy-Laufwerk 1541 ist kompatibel zu den anderen Commodore-Laufwerken mit der Bezeichnung CBM 4040 (Doppelfloppy für 4000er Serie) und CBM 2031 (Einzellaufwerk für den PET).

Wichtig ist, daß die Daten bis zum eigentlichen Schreiben auf der Diskette gepuffert werden, bis eine gewisse Anzahl von Bytes zur Verfügung steht. Deshalb ist auf jeden Fall in einem Programm nach Beendigung von Schreibarbeiten der CLOSE-Befehl zu geben, um alle Daten aus dem Puffer auch wirklich auf die Diskette zu übertragen.

Zum Befehlsvorrat ist zu sagen, daß nicht alle Befehle im Basic zur Verfügung stehen, sondern teilweise über den Kommandokanal (Kanal 15) übergeben werden müssen. Dies sind die Befehle NEW, VERIFY, RENAME, COPY und SCRATCH. Auch die Fehler- und Statusinformationen können über Kanal 15 abgefragt werden.

2/3.1.2

Das Laufwerk 1570

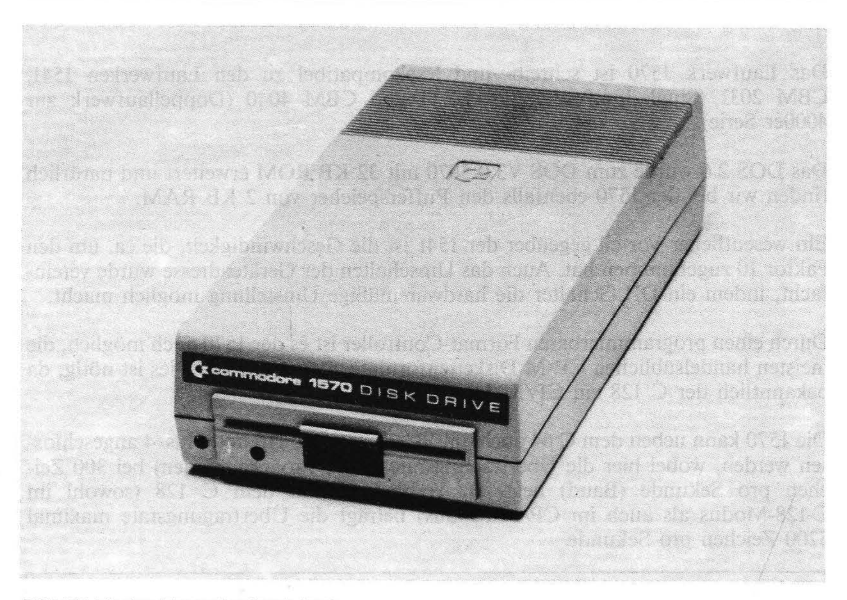


Bild 2/3.1.2 Das Floppy-Laufwerk 1570

Das Laufwerk 1570 ist eine Zwischenentwicklung zwischen dem Laufwerk 1541 und dem Laufwerk 1571. Die Laufwerke 1570 und 1571 sind hauptsächlich in Verbindung mit dem C 128 gedacht, wobei das Laufwerk 1570 alle Merkmale des 1571 aufweist, bis auf das neue Gehäuse und den zweiten Schreib-/Lesekopf.

Wahrscheinlich haben Materialengpässe die Firma Commodore veranlaßt, die Floppy 1570 im Vorgriff auf die 1571 zu produzieren.

2/3.1.2.1

Allgemeine Daten

Das Laufwerk 1570 ist schreib- und lesekompatibel zu den Laufwerken 1541, CBM 2031 (Single-Laufwerk zum PET) und CBM 4040 (Doppellaufwerk zur 4000er Serie).

Das DOS 2.6 wurde zum DOS V3.0 5070 mit 32 KB ROM erweitert und natürlich finden wir bei der 1570 ebenfalls den Pufferspeicher von 2 KB RAM.

Ein wesentlicher Vorteil gegeüber der 1541 ist die Geschwindigkeit, die ca. um den Faktor 10 zugenommen hat. Auch das Umschalten der Geräteadresse wurde vereinfacht, indem ein DIL-Schalter die hardwaremäßige Umstellung möglich macht.

Durch einen programmierbaren Format-Controller ist es der 1570 auch möglich, die meisten handelsüblichen CP/M Diskettenformate zu verarbeiten. Dies ist nötig, da bekanntlich der C 128 ein CP/M-Modus besitzt.

Die 1570 kann neben dem C 64 auch am VC 20, C 16, C 116 und plus/4 angeschlossen werden, wobei hier die Übertragungsrate (beim Programmladen) bei 300 Zeichen pro Sekunde (Baud) liegt. In Verbindung mit dem C 128 (sowohl im C-128-Modus als auch im CP/M-Modus) beträgt die Übertragungsrate maximal 5200 Zeichen pro Sekunde.

Technische Daten	
Schnittstelle	serieller IEEE-Bus (IEC)
Geräteadresse	8 (kann softwaremäßig oder über DIL-Schalter hardwaremäßig geändert werden)
GCR-Format (Group Code Recording)	
single sided/single density	
Kapazität	
Unformatiert	252019 Bytes
Formatiert	174848 Bytes
Größte sequentielle Datei	168656 Bytes
Größte relative Datei	167132 Bytes
Einträge pro relative Datei	65535
Dateien pro Diskette	144
Spuren auf der Diskette	35
Sektoren pro Spur	17-21
Sektoren pro Diskette	683 maximal 664 frei
Bytes pro Sektor	256

Technische Daten (Fortsetzung)
MFM-Format (Modifed Frequency Modulation)
single sided/double density

Kapazität		
Unformatiert	500000 Bytes	
Sektorgröße 128	133120 Bytes	
Sektorgröße 256	163840 Bytes	
Sektorgröße 512	184320 Bytes	
Sektorgröße 1024	204800 Bytes	
Maximale Anzahl der Spuren	40	
Sektoren pro Spur		
Sektorgröße 128	26	
Sektorgröße 256	16	
Sektorgröße 512	9	
Sektorgröße 1024	5	
Befehlsvorrat		
Files	OPEN	— Datei öffnen
	CLOSE	— Datei schließen
	LOAD	— Programm laden
	SAVE	— Programm sichern
	LOAD „$“	— Inhaltsverzeichnis laden
	NEW	— Diskinhalt löschen
	VERIFY	— Programm im RAM mit File vergleichen
	RENAME	— Datei umbenennen
	COPY	— Datei kopieren
	SCRATCH	— Datei löschen
Direktzugriff	BLOCK-READ	
	BLOCK-WRITE	
	BLOCK-EXECUTE	
	BLOCK-ALLOCATE	
	BLOCK-FREE	
	MEMORY READ	
	MEMORY WRITE	
	MEMORY-EXECUTE	
	BUFFER-POINTER	
	USER	
	BLOCK bezieht sich dabei direkt auf die Information auf der Diskette (Block=Sektor), MEMORY auf den Speicher des DOS	
	Fehler- und Statusinformationen können über einen eigenen Datenkanal abgefragt werden	
Filetypen	sequentiell, relativ (Direktzugriff/Random Access), Programmdateien, User-Dateien	
Aufzeichnungsverfahren		
Standard	GCR	
CP/M	GCR, MFM	
Schreib-/Lesekopf	1	
Netzspannung	220 V, 50 Hz	
Abmessungen		
Breite	200 mm	
Tiefe	374 mm	
Höhe	97 mm	
Gewicht	3,8 kp	

Mitgeliefertes Zubehör
Ausführliches Bedienungshandbuch, Test-/Demodiskette mit DOS-SHELL und Verbindungskabel zur Zentraleinheit.

Lassen Sie sich nicht von der Bezeichnung GCR-Format und MFM-Format bei den technischen Daten irritieren. Im GCR-Format finden Sie den bisherigen Commodore-Standard, wie er auch bei der 1541 vorhanden ist. Das MFM-Format beschreibt die Disketteneinteilung bei Verwendung von CP/M, wobei hier — nach Angabe von Commodore — auch eine Lese-/Schreibkompatibilität zu einem Kaypro II und einem Osborn DD besteht.

2/3.5 Joystick

In sehr vielen Fällen — nicht nur bei Spielprogrammen — erübrigt sich bei Verwendung eines Joysticks der Gebrauch der Tastatur. An anderer Stelle im Buch werden wir sogar den Joystick dazu heranziehen, eine Maus zu simulieren.

Angeschlossen werden die Joysticks entweder an Controlport 1 oder Controlport 2, wobei Controlport 2 in der Regel bevorzugt wird, da sich hier keine Komplikationen ergeben. Teilweise erfolgt die interne Behandlung der Joysticks analog mit der Tastatur, so daß sich hier insbesondere bei Controlport 1 Unstimmigkeiten zeigen können, worauf wir später aber noch eingehen werden.

Wenn Sie Ihr Anwenderprogramm mit einem Joystick steuern wollen, so können Sie die Stellung des Joysticks über die rechnerinternen Adressen 56320 (Controlport 2) oder 56321 (Controlport 1) erfragen. Hierzu wird der PEEK-Befehl verwendet. Da beide Adressen Ein-Byte-Register sind, können Werte von 0 bis 255 dargestellt werden. Die zugehörigen Dezimalzahlen zu den einzelnen Joystickstellungen an den verschiedenen Ports können Sie durch folgende Programmzeile auf den Bildschirm holen:

```
10 PRINT PEEK (56320) : GOTO 10
```

oder

```
10 PRINT PEEK (56321) : GOTO 10
```

Haben Sie dies durchgeführt, so ergeben sich die in den Abbildungen 2/3.5-1 bis 2/3.5-4 dargestellten Werte.

Die Werte sind im übrigen für den C 64 und C 128 identisch. Verwendet man beim C 128 jedoch den JOY-Befehl, so ergeben sich die in den Abbildungen 2/3.5-5 und 2/3.5-6 aufgeführten Werte.

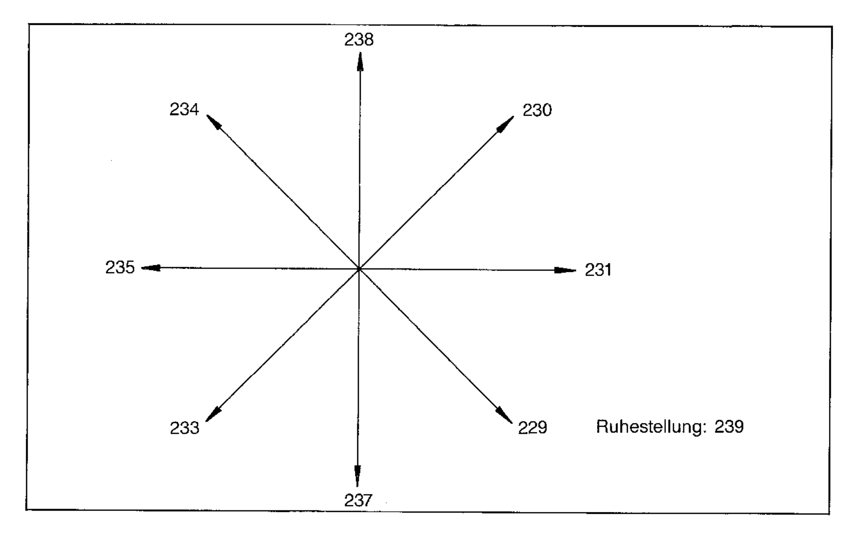


Bild 2/3.5-1 Joystickwerte an Control-Port 1 mit gedrückter Feuertaste

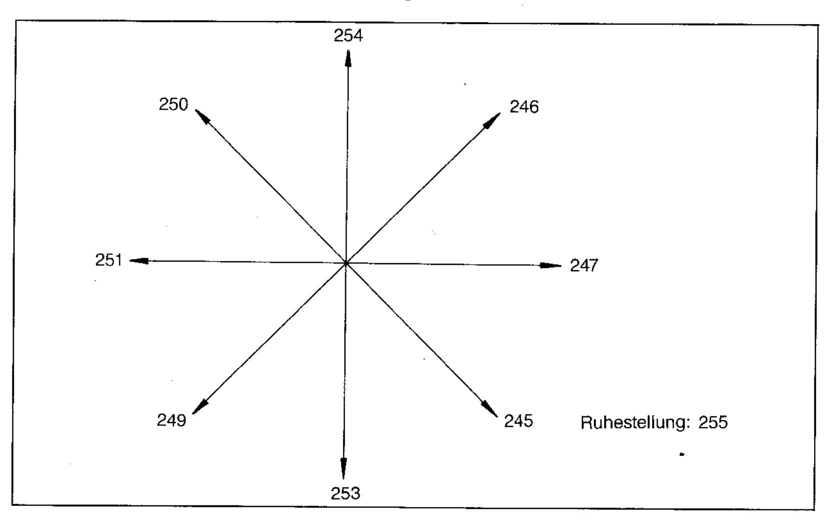


Bild 2/3.5-2 Joystickwerte am Control-Port 1 ohne Feuertaste

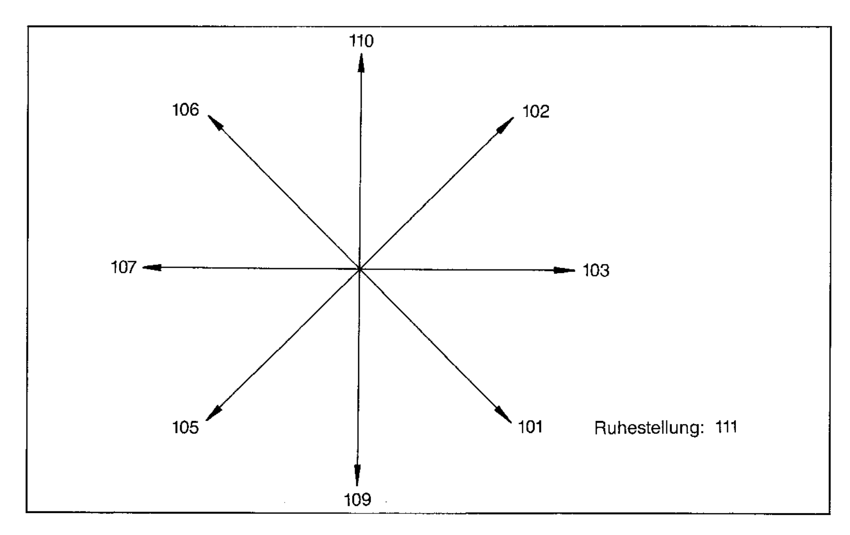


Bild 2/3.5-3 Joystickwerte an Control-Port 2 mit gedrückter Feuertaste

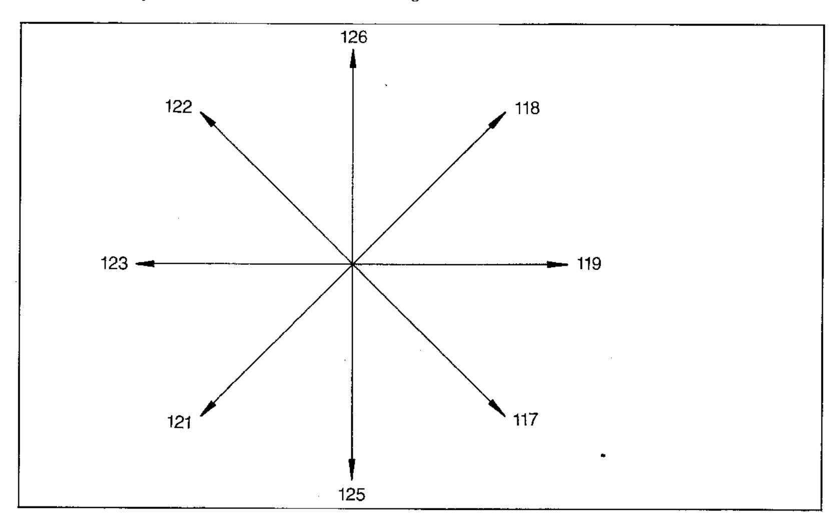


Bild 2/3.5-4 Joystickwerte am Control-Port 2 ohne Feuertaste

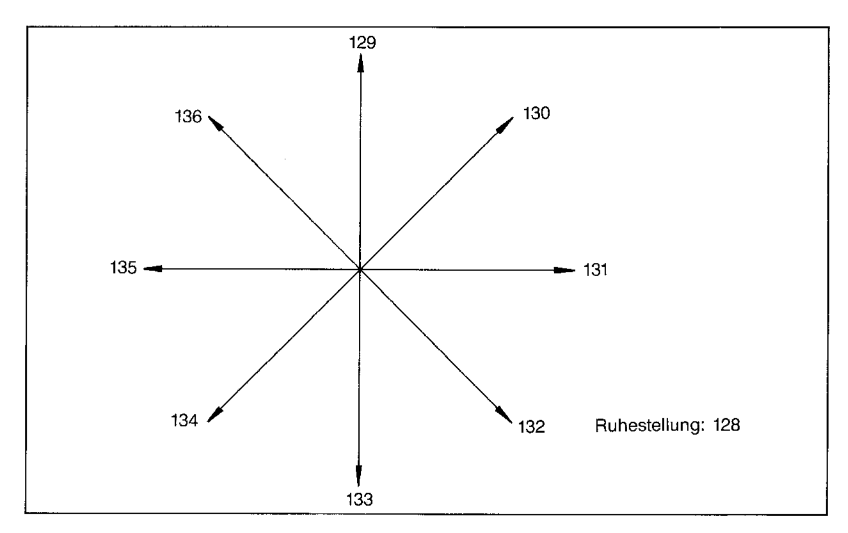


Bild 2/3.5-5 Joystickwerte bei Verwendung des JOY-Befehls im 128er-Modus mit gedrückter Feuertaste

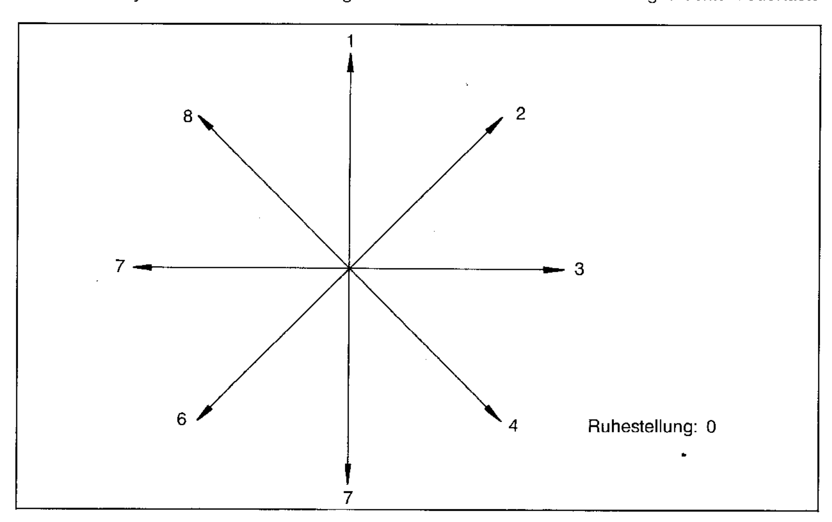


Bild 2/3.5-6 Joystickwerte bei Verwendung des JOY-Befehls im 128er-Modus ohne Feuertaste

Wenn man die Dezimalzahlen genauer interpretiert, so stellt man fest, daß die einzelnen Richtungen und der Feuerknopf jeweils durch ein Bit innerhalb der eingelesenen Bytes gesteuert werden. Die Diagonalen ergeben sich aus den Bitkombinationen der vier Hauptrichtungen, und auch dem Feuerknopf ist ein Bit zugeordnet. Näheres ist aus Bild 2/3.5-7 ersichtlich.

Bitmuster für Control-Port 2

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	1	Feuer-knopf 1	rechts	links	unten/ hinten	oben/ vorne
128	64	32	16	8	4	2	1

Richtung gilt, wenn aus „1" eine „0" wird.

Bitmuster für Control-Port 1

1	1	1	Feuer-knopf 1	rechts	links	unten/ hinten	oben/ vorne

Richtung gilt, wenn aus „1" eine „0" wird.

Bild 2/3.5-7 Bitmuster für Auswertung der Joystickabfrage an den Control-Ports mit PEEK. Bei JOY(n) erfolgt eine Umrechnung im Basic.

Bei Verwendung eines Joysticks sollte man den Wert der Abfrage zunächst einer Variablen (z.B. J0) zuweisen, bevor man im weiteren Programmablauf die abgefragte Richtung weiterverarbeitet. Geschieht dies nicht, so kann sich während der Verarbeitung — besonders bei Basic-Programmen — die Richtung und damit der zu verarbeitende Wert ändern. Dies ist in den meisten Fällen unerwünscht.

Im Folgenden wollen wir Ihnen ein kleines Rahmenprogramm vorstellen, das Sie bei der Verwendung eines Joysticks heranziehen können:

```
100 JO=PEEK(56320)
110 :
120 REM ----- OBEN RECHTS -----
130 IF (JO AND 9)=0 THEN ...
140 :
150 REM ----- UNTEN RECHTS -----
160 IF (JO AND 10)=0 THEN ...
170 :
180 REM ----- UNTEN LINKS -----
190 IF (JO AND 6)=0 THEN ...
200 :
210 REM ----- OBEN LINKS -----
220 IF (JO AND 5)=0 THEN ...
230 :
240 REM ----- OBEN -----
250 IF (JO AND 1)=0 THEN ...
260 :
270 REM ----- UNTEN -----
280 IF (JO AND 2)=0 THEN ...
290 :
300 REM ----- LINKS -----
310 IF (JO AND 4)=0 THEN ...
320 :
330 REM ----- RECHTS -----
340 IF (JO AND 8)=0 THEN ...
350 :
360 REM ----- FEUERKNOPF -----
370 IF (JO AND 16)=0 THEN ...
380 :
```

Listing 2/3.5-1: Rahmenprogramm bei Joystickverwendung

Wir haben die Vorgehensweise über die Bitmuster gewählt, um nicht jeweils die Richtung mit oder ohne Feuerknopf abfragen zu müssen. Dies geschah unter der Berücksichtigung der Tatsache, daß in der Regel die Feuertaste unabhängig von der gerade gewählten Richtung eine eigene Funktion hat. Neben Schießen kann dies z.B. auch die Auswahl eines angewählten Menüpunktes sein.

Es ist zu beachten, daß bei dieser Vorgehensweise zunächst die Diagonalen abgefragt werden. Würde man als erstes eine der vier Hauptrichtungen abfragen, so kämen die Diagonalen nie zum Zuge, da z.B. die Richtung rechts (Bit zum Dezimalwert 8) bereits eine Verzweigung bringen würde, auch wenn Bit 0 (Dezimal 1) gesetzt ist und dies insgesamt die Richtung „oben rechts" bedeuten würde.

An mehreren Stellen in diesem Buch werden wir den Joystick benutzen, so daß wir an dieser Stelle keine ausführlichen Beispiele bringen wollen.

Kollisionen mit der Tastatur

Wir hatten bereits gesagt, daß es beim Control-Port 1 zu Kollisionen mit der Tastatur kommen kann. Besonders schön deutlich wird es am C 128 mit folgendem kleinen Programm:

```
10 PRINT PEEK (56321) : GET A$ : PRINT A$ : GOTO 10
```

Wenn Sie dieses Miniprogramm starten und zunächst den Joystick in Ruhestellung belassen, so erscheint ein über die andere Zeile der Wert 255, was für die Kodierung der Ruhestellung auch richtig ist.

Wenn Sie nun den Feuerknopf kurz drücken, so erscheint buchstabenweise in den Zwischenräumen der Begriff MONITOR. Durch den Feuerknopf kann man also den Druck auf die Funktionstaste F8 simulieren. Auf die sich ändernden Zahlenwerte für die einzelnen Joystickpositionen wollen wir im Folgenden nicht mehr eingehen. Bewegen Sie jetzt den Joystick nach vorne, ändert sich außer dem Zahlenwert nichts. Sobald Sie jedoch nach vorne rechts gehen, verschwinden die Zwischenräume zwischen den Zahlwerten. Ebenso, wenn der Joystick nach rechts und unten rechts bewegt wird.

Bewegen Sie ihn nach unten links, links oder oben links, so erscheinen zwischen den Zahlen plötzlich zwei Zwischenräume. Drücken Sie bei den letztgenannten drei Richtungen zusätzlich den Feuerknopf so simulieren sie die Funktionstaste F7.

Besonders bei Anwenderprogrammen die mit Joystick und Tastatur arbeiten, kann es zu Problemen kommen. Arbeiten Sie z.B. zusätzlich zum Joystick mit einem Menü, dessen Wert mit dem GET-Befehl abgefragt wird, so kann es passieren, daß Sie nach Drücken des Feuerknopfes den zu M gehörigen Menüpunkt aufrufen, ohne jedoch die Taste M gedrückt zu haben. Das M rührt vom MONITOR-Befehl her.

Sofern also nur ein Joystick in Ihrem Anwenderprogramm verwendet werden soll, ist auf jeden Fall der Anschluß an Control-Port 2 vorzuziehen.

Teil 3

Interne Software

3/1
Das Betriebssystem des C 64

Im Rahmen der internen Software wollen wir zunächst auf die softwaremäßige Grundausstattung des C 64 eingehen: Ohne Betriebssystem läuft nichts. Das Betriebssystem gestattet z.B. dem Anwender die Eingabe von Zeichen auf der Tastatur, es sorgt für die Darstellung der Zeichen auf dem Bildschirm, es verwaltet den internen Speicher und auch die Befehle im Zusammenhang mit dem Kassettenrekorder oder einer Floppystation sind begrifflich zum Betriebssystem zu zählen, obwohl sie vom Basic in ihrer Anwendung nicht zu unterscheiden sind. Kurz und gut, das Betriebssystem ist das Kernstück der Anlage, das die einzelnen Komponenten sinnvoll zusammenfügt.

Im Rahmen der Beschreibung des Betriebssystems werden wir zunächst eine Übersicht über die Speicherbelegung des C 64 mit RAM und ROM bringen. Diese sind eine Grundvoraussetzung für das Verständnis der weiteren Teile.

Als nächstes folgt eine Übersicht über die wichtigsten Adressen, wobei wir zunächst den Variablenbereich, den das Betriebssystem im ersten KByte das RAM belegt, besprechen wollen. Neben einer Kurzübersicht wird jede einzelne Adresse ausführlich besprochen.

Im Rahmen des Betriebssystems wollen wir auch die Register der einzelnen Zusatzbausteine (VIC, SID und CIA) aufzeigen, um die Adreßübersicht abzurunden.

3/1.1
Übersicht und Speicherbelegung

Mit einem 8-Bit-Prozessor, wie es der 6510 im C 64 ist, kann man insgesamt 65536 Speicherzellen adressieren, da zur Adressierung zwei Byte zur Verfügung stehen. Nun hat der C 64 seinen Namen von den 64 KByte-RAM, die in ihm enthalten sind.

Da aber außerdem noch ein Betriebssystem und eine Sprache (der Basic-Interpreter) gebraucht werden, weiterhin der Zeichensatz untergebracht werden und ein Adreßbereich für Ein-/Ausgabeoperationen vorhanden sein muß, gibt es ein paar kleine Probleme, auf die wir im Rahmen dieses Kapitels eingehen wollen.

Das Betriebssystem, der Basic-Interpreter und der Zeichensatzgenerator sind in ROM's (**R**ead **O**nly **M**emory — Nur-Lese-Speicher) untergebracht. Diese sind adressenmäßig in den Bereich von 0 bis 65535 einbezogen. Da man jedoch nur entweder RAM oder ROM adressieren kann, muß in den doppelt und teilweise sogar dreifach belegten Bereichen eine Auswahl getroffen werden, die im Register des Prozessor-Ports (Zweites Byte im RAM, siehe Kapitel 3/1.2.1.2) kodiert werden. Mit POKE 1,X (auf die Werte von X werden wir später noch eingehen) lassen sich die verschiedenen Speicherkonfigurationen erreichen.

Außerdem sind die Anschlüsse GAME und EXROM zu berücksichtigen, die vom Expansion-Port beeinflußt werden. Weiterhin ist an der Speicherkonfiguration noch der Grafikprozessor VIC beteiligt, auf den wir auch später eingehen wollen.

Zunächst wollen wir uns auf die durch den Prozessor-Port beeinflußbare Möglichkeiten beschränken.

In den Bildern 3/1.1-1 bis 3/1.1-7 haben wir die sieben möglichen Zustände für Sie grafisch aufbereitet. Durch unterschiedliche Schraffuren ist jeweils kenntlich gemacht, wo mittels des PEEK-Befehls gelesen werden kann, und wo der POKE-Befehl schreibend zur Anwendung kommt.

Da beim Prozessor-Port auch die Steuerung des Kassettenrekordermotors untergebracht ist, sind diese Bits zu berücksichtigen. Der Wert des Registers 1 hat nach dem Einschalten den Wert 55, wovon der Wert 48 für die Steuerung des Kassettenmotors generell zuständig ist. Durch Poken der Werte 48 bis 55 in Adresse 1, lassen sich die aufgezeigten Zustände darstellen.

Die daraus resultierenden Werte für die Bits 0 bis 2 dieser Adresse sind aus nachfolgender Tabelle ersichtlich, wobei aus nmemotechnischen Gründen jedem Bit noch ein Name gegeben wurde (CHAREN: Bit 2; HIRAM: Bit 1; LORAM: Bit 0):

Bit 210	POKE-Wert	Bedeutung	Darstellung in Bild
111	55	Einschaltbelegung	3/1.1-1
110	54	Basic-ROM ausblenden	3/1.1-2
101	53	Basic- u. Kernal-ROM ausblenden	3/1.1-3
100	52	RAM	3/1.1-4
000	48	RAM	3/1.1-4
011	51	Zeichengenerator einlesen	3/1.1-5
010	50	Zeichengenerator lesen u. Basic-ROM ausschalten	3/1.1-6
001	49	Nur RAM außer Zeichengenerator	3/1.1-7

Auffällig ist, daß der Wert von CHAREN unbedeutend ist, wenn sowohl HIRAM als auch LORAM auf 0 gesetzt sind.

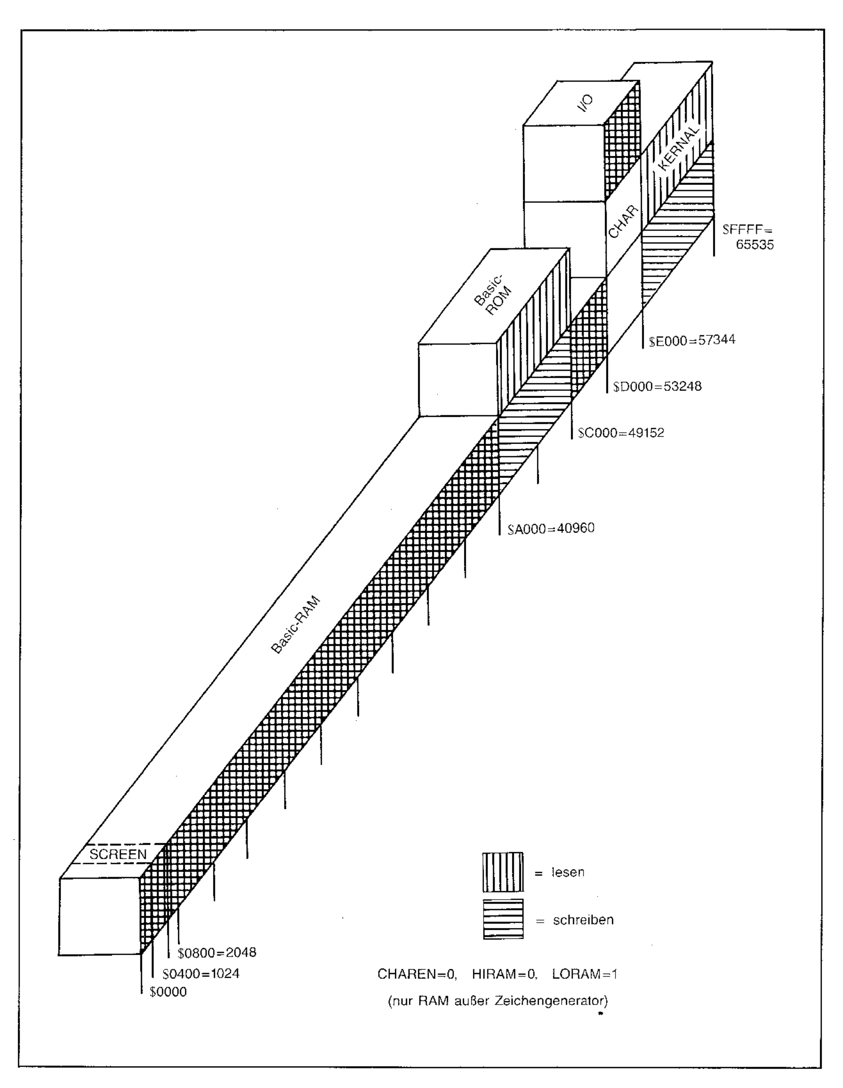

I/O
KERNAL
CHAR
$FFFF=
65535
Basic-
ROM
$E000=57344
$D000=53248
$C000=49152
$A000=40960
Basic-RAM
SCREEN
= lesen
= schreiben
$0800=2048
$0400=1024
$0000
CHAREN=0, HIRAM=0, LORAM=1
(nur RAM außer Zeichengenerator)

Bild 3/1.1-1

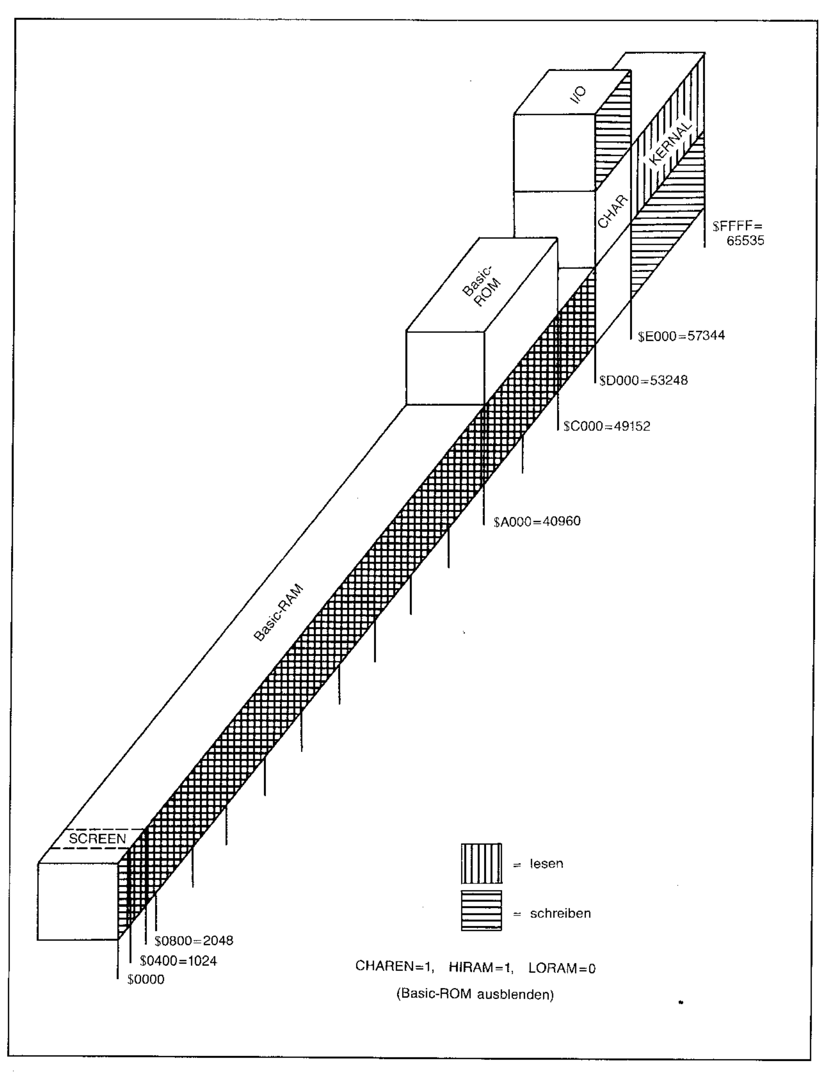

I/O
KERNAL
CHAR
$FFFF=
65535
Basic-
ROM
$E000=57344
$D000=53248
$C000=49152
$A000=40960
Basic-RAM
SCREEN
= lesen
= schreiben
$0800=2048
$0400=1024
$0000
CHAREN=1, HIRAM=1, LORAM=0
(Basic-ROM ausblenden)

Bild 3/1.1-2

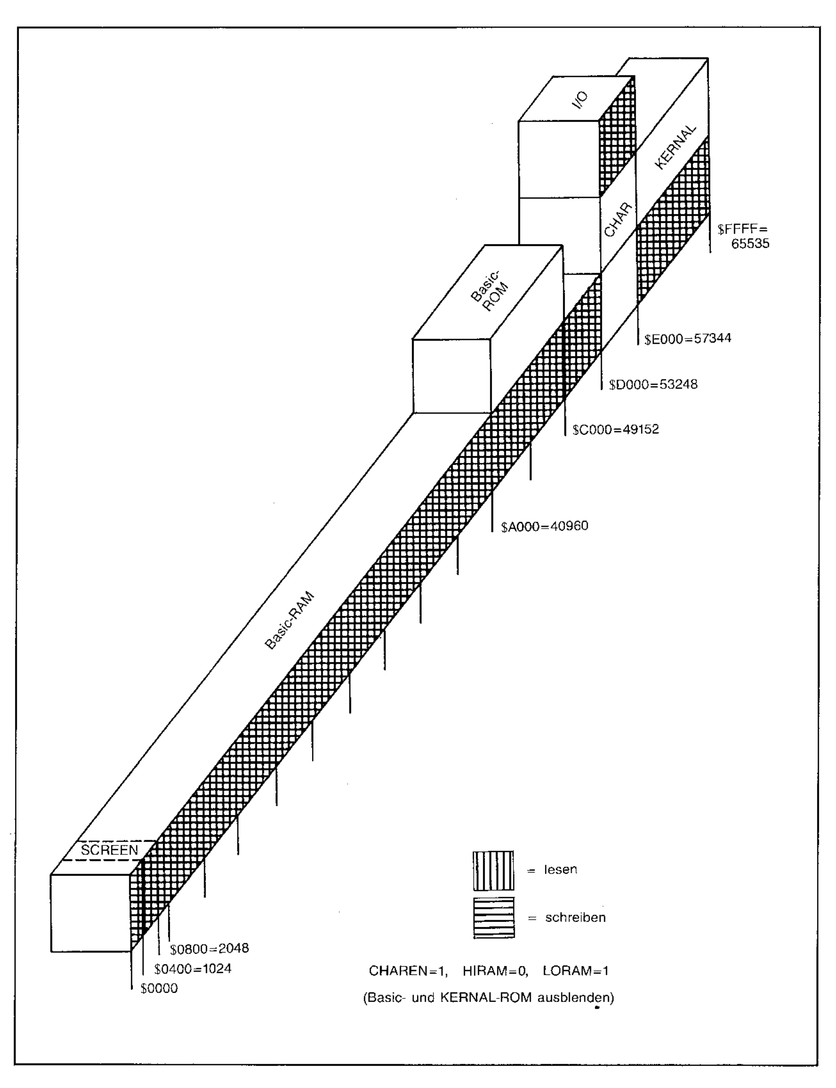

I/O
KERNAL
CHAR
$FFFF=
65535
Basic-
ROM
$E000=57344
$D000=53248
$C000=49152
$A000=40960
Basic-RAM
SCREEN
= lesen
= schreiben
$0800=2048
$0400=1024
$0000
CHAREN=1, HIRAM=0, LORAM=1
(Basic- und KERNAL-ROM ausblenden)

Bild 3/1.1-3

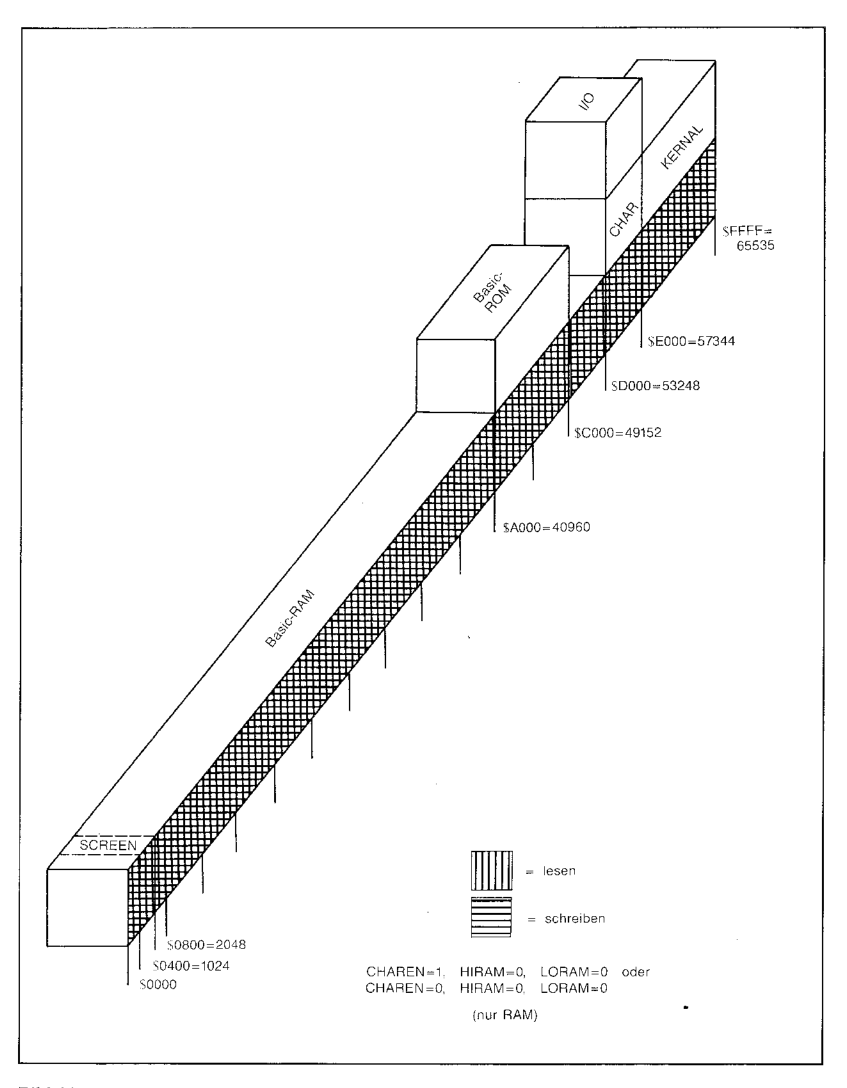

I/O
KERNAL
CHAR
$FFFF=
65535
Basic-
ROM
$E000=57344
$D000=53248
$C000=49152
$A000=40960
Basic-RAM
SCREEN
= lesen
= schreiben
$0800=2048
$0400=1024
$0000
CHAREN=1, HIRAM=0, LORAM=0 oder
CHAREN=0, HIRAM=0, LORAM=0
(nur RAM)

Bild 3/1.1-4

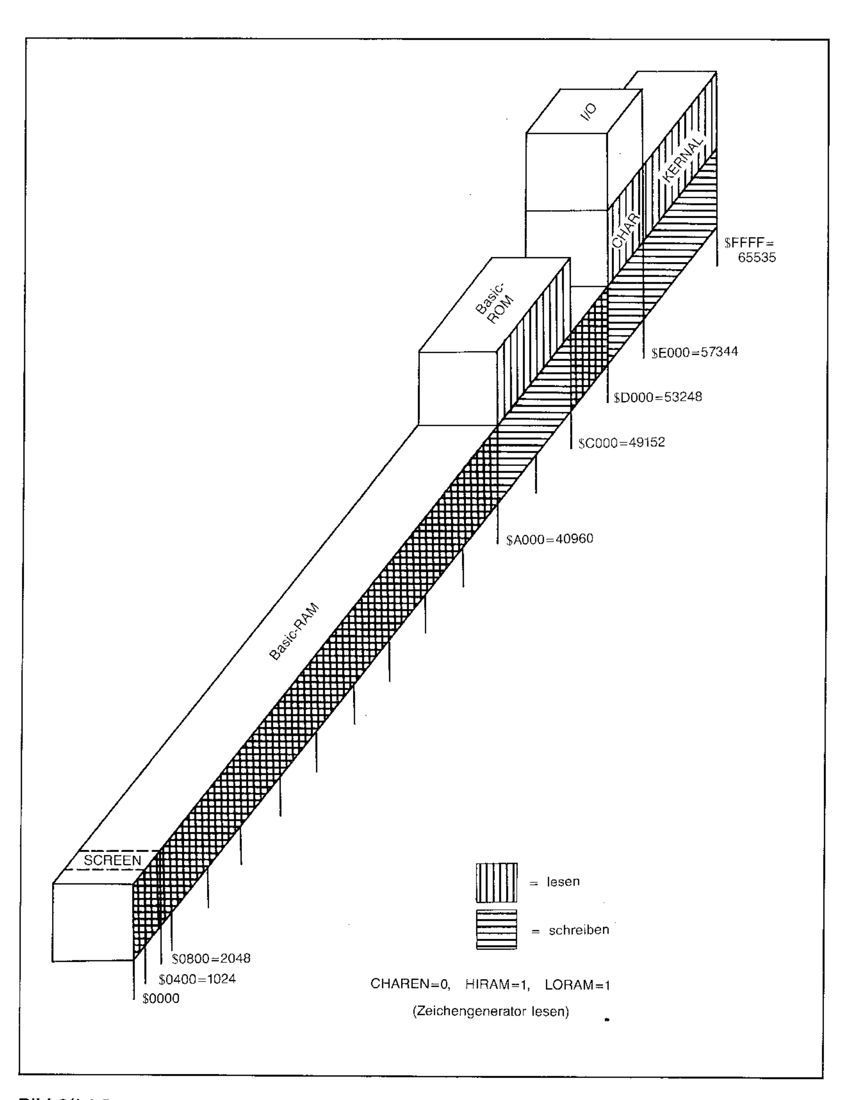

I/O
KERNAL
CHAR
$FFFF=
65535
Basic-
ROM
$E000=57344
$D000=53248
$C000=49152
$A000=40960
Basic-RAM
SCREEN
= lesen
= schreiben
$0800=2048
$0400=1024
$0000
CHAREN=0, HIRAM=1, LORAM=1
(Zeichengenerator lesen)

Bild 3/1.1-5

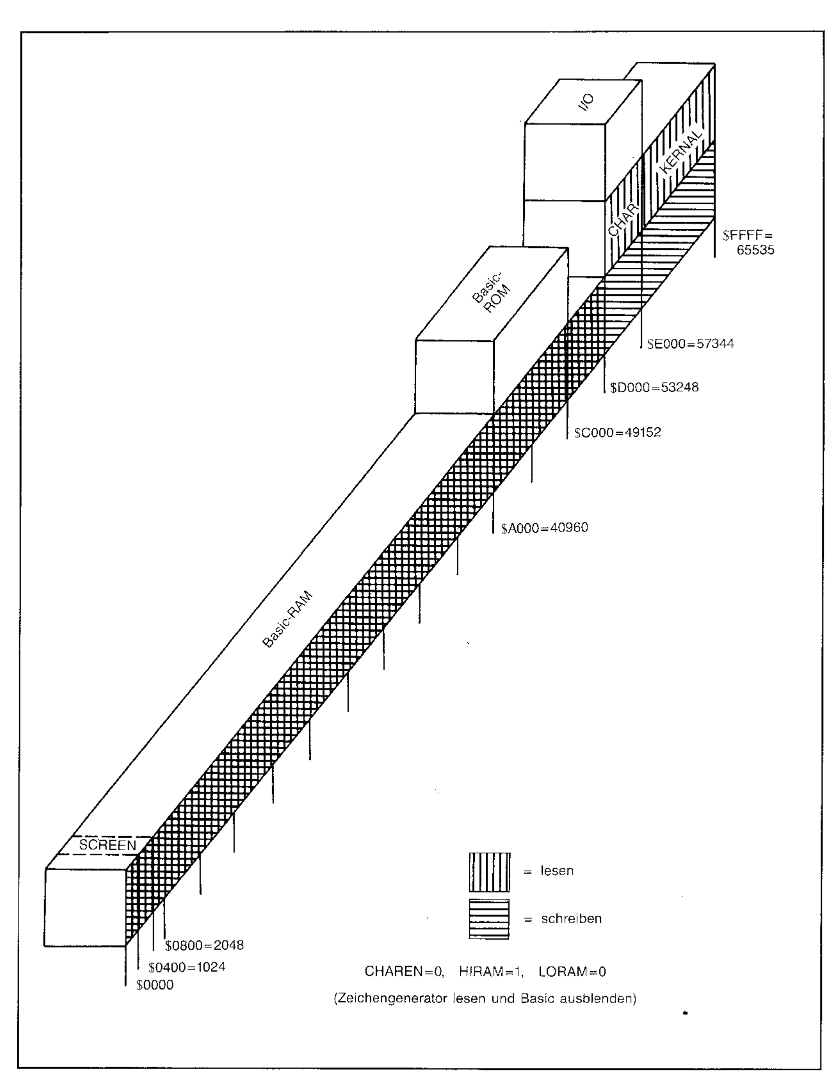

I/O
KERNAL
CHAR
$FFFF=
65535
Basic-
ROM
$E000=57344
$D000=53248
$C000=49152
$A000=40960
Basic-RAM
SCREEN
= lesen
= schreiben
$0800=2048
$0400=1024
$0000
CHAREN=0, HIRAM=1, LORAM=0
(Zeichengenerator lesen und Basic ausblenden)

Bild 3/1.1-6

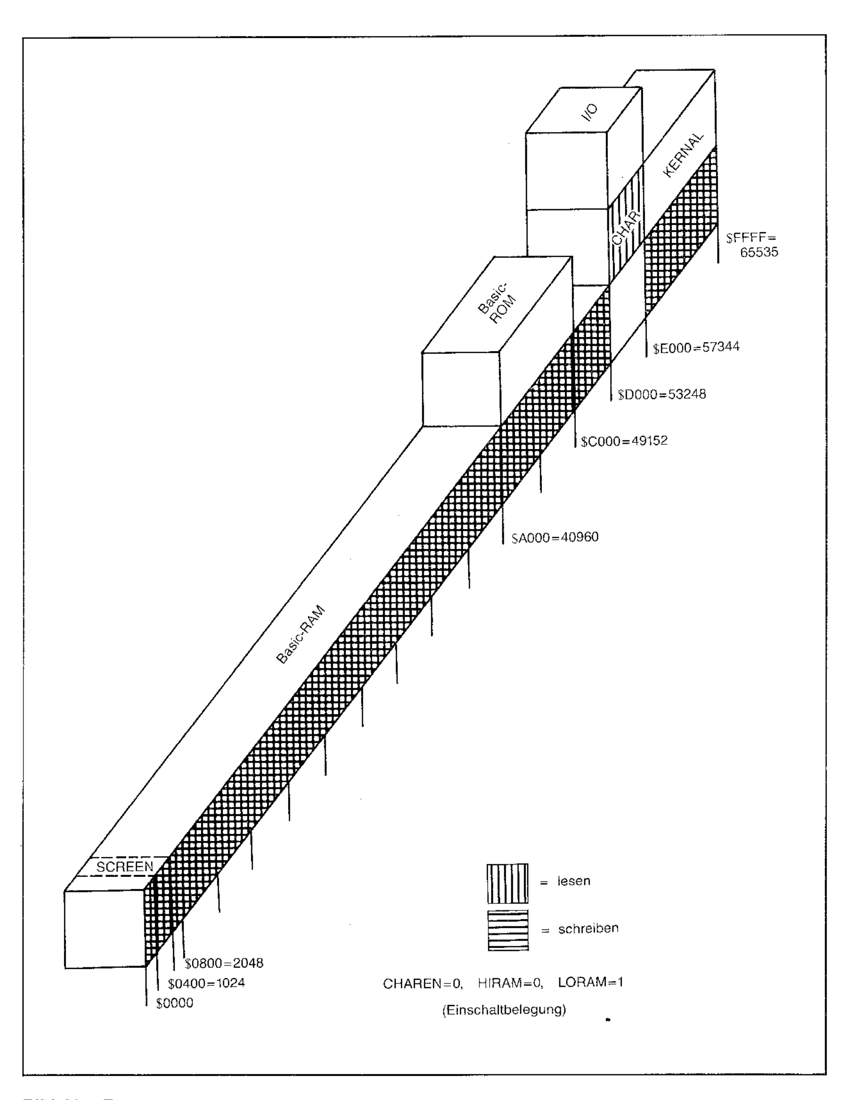

I/O
KERNAL
CHAR
Basic-ROM
Basic-RAM
SCREEN
$FFFF=65535
$E000=57344
$D000=53248
$C000=49152
$A000=40960
$0800=2048
$0400=1024
$0000
= lesen
= schreiben
CHAREN=0, HIRAM=0, LORAM=1
(Einschaltbelegung)

Bild 3/1.1-7

3/1.2

Die wichtigsten Adressen

Im Folgenden haben wir für Sie die wichtigsten Adressen des C 64 in tabellarischer Form zusammengestellt. Es handelt sich dabei in der Hauptsache um die Zero-Page (Adressen 0 bis 255 im RAM) und die folgenden Adressen bis 1023. Der Vollständigkeitshalber wollen wir auch die Register der einzelnen Bausteine ebenfalls angeben.

3/1.2.1

Zero-Page

Um besser nachschlagen zu können, bringen wir zuerst alle Adressen der Zero-Page in einer Gesamtübersicht. Meist weiß man ungefähr, wo man suchen muß, nur einige Details sind noch nachzuschlagen. Benutzen Sie hierzu Kapitel 3/1.2.1.1.

Wer sich als erstes einmal mit der Wirkungsweise der Adressen vertraut machen möchte, findet in Kapitel 3/1.2.1.2 eine genaue Aufstellung aller Adressen, versehen mit einem ausführlichen Kommentar und gelegentlich weiterführende Hinweise, sowie Tips und Tricks.

3/1.2.1.1

Übersicht

Hexadez.	Dezimal	Belegung
00	0	Datenrichtungsregister für Prozessorport
01	1	Prozessorport
02 - 06	2 - 6	frei
07	7	Suchzeichen
08	8	Merker für Hochkomma-Modus
09	9	Speicher für Spalte beim TAB-Befehl
0A	10	Load - 0; Verify - 1
0B	11	Zeiger in Eingabepuffer, Anzahl der Dimensionen
0C	12	Merker: für DIM
0D	13	Merker: $00-numerisch, $FF-String
0E	14	Merker: $80-Integer, $00-Fließkomma
0F	15	Merker: für Hochkomma-Modus bei LIST
10	16	Merker: für FN
11	17	Merker: INPUT-$00, GET-$40, READ-$98
12	18	Vorzeichen bei ATN
13	19	aktives I/0-Gerät
14 - 15	20 - 21	Integer-Adresse, z.B. Zeilennummer
16	22	Zeiger auf Stringstack
17 - 18	23 - 24	Zeiger auf zuletzt verwendeten String
19 - 21	25 - 33	Stringstack
22 - 23	34 - 35	Zeiger auf String nach FRESTR
24 - 25	36 - 37	Zeiger für diverse Zwecke
26 - 2A	38 - 42	Register für Funktionsauswertung und Arithmetik
2B - 2C	43 - 44	Zeiger auf Basic-Programmstart
2D - 2E	45 - 46	Zeiger auf Start der Variablen
2F - 30	47 - 48	Zeiger auf Start der Arrays
31 - 32	49 - 50	Zeiger auf Ende der Arrays
33 - 34	51 - 52	Zeiger auf Beginn der Strings
35 - 36	53 - 54	Hilfszeiger auf String
37 - 38	55 - 56	Zeiger auf BASIC-RAM Ende
39 - 3A	57 - 58	aktuelle BASIC-Zeilennummer
3B - 3C	59 - 60	letzte Basic-Zeilennummer
3D - 3E	61 - 62	Zeiger auf nächsten BASIC-Befehl für CONT
3F - 40	63 - 64	aktuelle Zeilennummer für DATA
41 - 42	65 - 66	Zeiger auf nächstes DATA-Element
43 - 44	67 - 68	Zeiger auf Eingabeelement
45 - 46	69 - 70	aktueller Variablenname
47 - 48	71 - 72	aktuelle Variablenadresse
49 - 48	73 - 74	Zeiger auf Variablenwert
4B - 4C	75 - 76	Zwischenspeicher für Programmzeiger
4D	77	Maske für Vergleichoperationen
4E - 4F	78 - 79	Zeiger für FN
50 - 53	80 - 83	Stringdescriptor

Hexadez.	Dezimal	Belegung
54	84	Konstante $4C JMP für Funktionen
55 - 56	85- 86	Sprungvektor für Funktionen
57 - 5B	87- 91	Register für Arithmetik, Akku # 3
5C - 60	92- 96	Register für Arithmetik, Akku # 4
61 - 65	97-101	Fließkommaakku # 1, FAC
66	102	Vorzeichen von FAC
67	103	Zähler für Polynomauswertung
68	104	Rundungsbyte für FAC
69 - 6D	105-109	Fließkommaakku # 2, ARG
6E	110	Vorzeichen von ARG
6F	111	Vergleichsbyte der Vorzeichen von FAC und ARG
70	112	Rundungsbyte für FAC
71 - 72	113-114	Zeiger für Polynomauswertung
73 - 8A	115-138	CHRGET-Routine, holt Zeichen aus BASIC-Text
79	121	CHRGOT-Routine
7A - 7B	122-123	Programmzeiger
8B - 8F	139-143	letzter RND-Wert
90	144	Statuswert ST
91	145	Merker für STOP-Taste
92	146	Zeitkonstante für Band
93	147	Merker für LOAD-$00 oder VERIFY-$01
94	148	Merker bei IEC-Ausgabe
95	149	Ausgabepuffer für IEC-Bus
96	150	Merker für EOT vom Band empfangen
97	151	Zwischenspeicher für Register
98	152	Anzahl der offenen Dateien
99	153	aktuelles Eingabegerät
9A	154	aktuelles Ausgabegerät
9B	155	Parität für Band
9C	156	Merker für Byte empfangen
9D	157	Merker für Direkt-Modus-$80, Programm-$00
9E	158	Band Paß 1 Prüfsumme
9F	159	Band Paß 2 Fehlerkorrektur
A0 - A2	160-162	Uhr (TI)
A3	163	Bitzähler für serielle Ausgabe
A4	164	Zähler für Band
A5	165	Zähler für Band schreiben
A6	166	Zeiger in Bandpuffer
A7 - A8	167-171	Arbeitsspeicher für Bandein/ausgabe
AC - AD	172-173	Zeiger für Bandpuffer und Scrolling
AE - AF	174-175	Zeiger auf Programmende bei LOAD/SAVE
B0 - B1	176-177	Zeitkonstanten für Band-Timing
B2 - B3	178-179	Zeiger auf Bandpuffer
B4	180	Bitzähler für Band
B5	181	nächstes Bit für RS 232
B6	182	Puffer für auszugebendes Byte
B7	183	Länge des Dateinames
B8	184	aktuelle logische Dateinummer
B9	185	aktuelle Sekundäradresse
BA	186	aktuelle Gerätenummer
BB - BC	187-188	Zeiger auf Dateinamen
BD	189	serielle Ein/Ausgabe
BE	190	Paßzähler für Band
BF	191	Puffer für serielle Ausgabe

Hexadez.	Dezimal	Belegung
C0	192	Merker für Bandmotor
C1 - C2	193-194	Startadresse für Ein/Ausgabe vom Bildschirm
C3 - C4	195-196	Endadresse für Ein/Ausgabe vom Bildschirm
C5	197	Nummer der gedrückten Taste, 64: keine Taste
C6	198	Anzahl der gültigen Zeichen im Tastaturpuffer
C7	199	Merker für RVS-Modus
C8	200	Zeilenende für Eingabe
C9	201	Cursorzeile für Eingabe
CA	202	Cursorspalte für Eingabe
CB	203	gedrückte Taste, 64: keine Taste
CC	204	Merker: Cursor ein-0; Cursor aus-1
CD	205	Zähler für Cursor blinken
CE	206	Zeichen unter dem Cursor
CF	207	Merker für Cursor, Ein-Phase - 1; Aus-Phase - 0
D0	208	Merker für Eingabe von Tastatur oder Bildschirm
D1 - D2	209-210	Zeiger auf Start der aktuellen Bildschirmzeile
D3	211	Cursorspalte
D4	212	Merker für Hochkommamodus
D5	213	Länge der Bildschirmzeile
D6	214	Cursorzeile
D7	215	für verschiedene Zwecke
D8	216	Anzahl der Inserts
D9 - F2	217-242	MSB der Bildschirmzeilenanfänge
F3 - F4	243-244	Zeiger in Farb-RAM
F5 - F6	245-246	Zeiger auf Tastatur-Dekodiertabelle
F7 - F8	247-248	Zeiger auf RS 232 Eingabepuffer
F9 - FA	249-250	Zeiger auf RS 232 Ausgabepuffer
FB - FE	251-254	unbenutzt

3/1.2.1.2

Zero-Page — Kommentar und Tips

Die Zero-Page ist nicht nur einer der wichtigsten Speicherbereiche in Ihrem C 64, sondern auch der interessanteste. Jeder wichtige Ablauf benutzt irgendwann einmal die Zero-Page. Ohne Zero-Page könnten Betriebssysteme und Assembler nicht arbeiten.

Profis, die sich mit dieser Materie schon beschäftigt haben, werden wohl am häufigsten auf die bereits abgedruckte Übersicht zurückgreifen. Trotzdem gibt es einige Zusammenhänge, die aus dieser Übersicht nicht klar hervorgehen. Zum einen aus

diesem Grund, zum anderen, um den Fortgeschrittenen und anderweitig Interessierten einen Einblick in die Arbeitsweise des Rechners zu geben, werden wir zunächst die Zero-Page und in Kapitel 3/1.2.2.2 die Adressen 256 bis 1023 näher beleuchten.

Adreßdarstellung in zwei Byte

In diesem Bereich kommt es häufig vor, daß zwei Byte zu einer Adreßangabe oder einem Wert mit anderer Funktion zusammengefaßt werden. Dadurch lassen sich Zahlen bis 65535 darstellen. Der Konvention zur Folge stellt das erste Byte dabei den niedrigeren Teil dar und das zweite Byte den höherwertigen. Allgemein wird hier auch von Low Byte und High Byte gesprochen. Die durch beide Bytes dargestellte Zahl läßt sich durch folgende Formel sehr leicht errechnen:

Zahl = High Byte * 256 + Low Byte.

Wollen Sie einen Wert zwischen 256 und 65535 in zwei Bytes unterbringen, so ist als Umkehrung zunächst die Zahl durch 256 zu dividieren. Dieser Wert wird dann im High Byte abgelegt. Sofern Sie mit Hand dividiert haben, können Sie den Rest automatisch in das Low Byte deponieren. Bei Verwendung eines Taschenrechners sind die Nachkommastellen nochmals mit 256 zu multiplizieren, um den Wert des Low Bytes zu ermitteln.

Sinn der Zero-Page

Auch das Betriebssystem und der Basic-Interpreter, die als Programm im ROM gespeichert sind, brauchen Daten, die naturgemäß nicht im ROM, sondern im RAM untergebracht werden müssen. Deshalb ist das erste KByte im RAM des C 64 für die Variablen des Betriebssystems und des Basic-Interpreters vorgesehen.

Die Zero-Page ist hierbei besonders wichtig, da die Adressen 0 bis 255 unter Zuhilfenahme eines einzigen Bytes angesprochen werden können.

0 — $0000

Datenrichtungs-Register für Mikro-Prozessor

Der 6510 des C 64 hat sechs Ein-/Ausgabe-Leitungen, die einzeln programmiert werden können. Die Bits 0 bis 5 im Datenrichtungs-Register dienen nun dazu, für jede einzelne Leitung ihre Funktion als Eingang (Bit=0) oder Ausgang (Bit=1) festzulegen.

Nach dem Einschalten steht in diesem Register 47 entsprechend der Bitkombination xx101111, d.h., daß die Leitungen 1, 2, 3, 4, 6 auf Ausgang geschaltet sind, und nur die Leitung 5 als Eingang verwendet wird.

Die Programmierung des ersten Bytes in der Zero-Page bleibt voll und ganz dem Anwender überlassen, da das Betriebssystem und auch der Basic-Interpreter des C 64 nicht darauf zugreifen. Im Zusammenhang mit dem Datenrichtungs-Register

steht das Datenregister im zweiten Byte der Zero-Page. Hierzu sollte man auch die Übersicht in Kapitel 3/1.1 heranziehen.

1 **$0001**

Prozessorport (Datenregister des 6510)

Bei den Speicherbelegungen haben wir gesehen, daß die adressierbaren 64 KByte des Mikroprozessors mehrfach ausgenutzt werden. Neben den 64 KB RAM besitzt der 64er ja noch 24 KB ROM, die dem RAM überlagert sind. Dies im Bereich des Character-ROM sogar zweifach. Irgendwo muß nun beim Zugriff auf die doppelbelegten Adressen festgelegt werden, ob das RAM gelesen oder geschrieben werden soll, das Betriebssystem (Kernal) oder das Basic-Interpreter-ROM, das I/0 oder das Character-ROM auszulesen ist. Diese Funktion übernimmt das Datenregister in Speicherzelle 1.

Die Adresse ist insbesondere auch für Maschinenspracheprogrammierer wichtig, z.B. wenn er den Basic-Interpreter nicht braucht, und ihn ausschalten möchte. Besprechen wir jedoch die Funktion der Bits im einzelnen.

Bit 0 Bit 0 ist für die Umschaltung des Speicherbereiches von $A000 bis $BFFF (40960 bis 49151) zuständig. Aufgrund der Speicherbelegungspläne wissen wir, daß in diesem Bereich das Basic-ROM untergebracht ist. Bit 0 wird auch als LORAM bezeichnet. Nach dem Einschalten ist dieses Bit auf 1 gesetzt, was bedeutet, daß ein POKE auf den Adreßbereich $A000 bis $BFFF auf das RAM wirkt, also der entsprechende Wert dort eingeschrieben wird, ein PEEK-Befehl jedoch auf das Basic-ROM wirkt, also dort der entsprechende Befehl ausgelesen wird.

Wird Bit 0 auf 0 gesetzt, so wirken sowohl Schreib- als auch Leseoperationen im genannten Adreßbereich auf das dort untergebrachte RAM.

Bit 1 Bit 1 ist für den Adreßbereich $E000 bis $FFFF (57344 bis 65535) zuständig. Laut unseren Speicherbelegungsplänen ist dort das Betriebssystem, auch Kernal genannt, untergebracht. Hier ist nach dem Einschalten eine 1 vorhanden. Dieses Bit wird im allgemeinen auch als HIRAM bezeichnet. Genau wie bei Bit 0 wird bei einer eingetragenen 1 aus dem ROM gelesen und ins RAM geschrieben und bei einer eingetragenen 0 jeder Zugriff auf den entsprechenden RAM-Bereich durchgeführt.

Bit 2 Die gleiche Funktion wie die Bits 0 und 1 erfüllt auch Bit 2. Bit 2 ist allerdings für den Adreßbereich von $D000 bis $DFFF (53248 bis 57344) zuständig. Dieser Adreßbereich ist neben dem RAM doppelt mit ROM belegt, nämlich dem I/0-ROM (Ein-/Ausgabe) und dem Zeichengenerator (Character-ROM). In diesem Falle schaltet das Bit (Einschaltzustand 1) nicht zwischen dem normalen ROM und den beiden anderen Bereichen um, sondern zwischen I/0 und dem Zeichensatz. Die Bezeichnung für Bit 2 lautet

Wolken fractale Gebilde sind und ihr Aussehen anderen mathematischen Gesetzmäßigkeiten unterliegt: sie sind selbstähnlich.

Wie lang ist die Küstenlinie Großbritanniens?

In den folgenden Kapiteln wollen wir Sie nach und nach in das Thema der fractalen Strukturen einführen. Die erste bekannte Arbeit zu diesem Thema stammt aus dem Jahre 1967, von Benoit B. Mandelbrot und trug den Namen: „Wie lang ist die Küste Großbritanniens?" Mandelbrot wollte dabei natürlich nicht die genaue Länge der Küste Großbritanniens ermitteln, sondern er stellte ein paar grundlegende Tatsachen an diesem Beispiel fest, die wir nun kurz nachvollziehen wollen.

Landkarten in verschiedenen Maßstäben

Schauen Sie sich in verschiedenen Atlanten Karten von Großbritannien in unterschiedlichen Maßstäben an und fahren Sie vielleicht mit einem Kartenroller die Küstenlinie ab. Sie werden dabei eine interessante Entdeckung machen. Oder folgen Sie einfach einem abstrakten Beispiel: Nehmen Sie an, Sie sehen die Britische Insel aus 1000 km Höhe. Sie erkennen nur die groben Umrisse dieser Insel, besondere Feinheiten sind nicht auszumachen. Gehen sie auf 500 km Höhe hinunter, so verfeinert sich die Küstenlinie zusehends. Die Küstenlinie, die Sie sehen, wird länger. Was bei 1000 km noch eine gerade Linie ist, sieht nun nicht mehr so gerade aus. Wenn Sie weiter hinab gehen, z.B. auf 100 km, werden weitere Feinheiten in der Küstenlinie erkennbar und wenn sie jedesmal die Küstenlinie nachzeichnen, wird der Strich länger und länger. Neue, kleinere Buchten werden sichtbar. Dieses Beispiel kann man soweit treiben, daß die Küstenlinie Großbritanniens unendlich lang ist, obwohl bei einer Größenordnung von Atomen wohl nicht mehr von einer Küstenlinie gesprochen werden kann.

In den nachfolgenden Kapiteln wollen wir dies an einem sehr einfachen Beispiel verdeutlichen.

CHAREN als Abkürzung für CHARacter ENable. Auf den Zeichengenerator wird zugegriffen, wenn dieses Bit auf 0 gesetzt ist.

Mit den Bits 0 bis 2 lassen sich alle möglichen Konfigurationen zwischen RAM und ROM darstellen.

Die Bits 3 bis 5 beschäftigen sich mit dem Kassettenrekorder und haben folgende Bedeutung:

Bit 3 weist den Einschaltzustand 0 auf und ist für das Senden von seriellen Daten zum Kassettenrekorder zuständig.

Bit 4 weist den Normalzustand 1 aus und dient zur Prüfung der Tasten des Kassettenrekorders, und hier insbesondere der Tasten PLAY, REWIND und FFWD, da beim Einschalten dieser Tasten der Motor in Bewegung gesetzt werden muß, was durch Bit 5 erledigt wird.

Bit 5 Ist für den Schaltzustand des Motors verantwortlich (Einschaltbelegung 1).

Addiert man die oben beschriebenen Einschaltzustände der einzelnen Bits, so ergibt dies den Dezimalwert 55. Die Bits zur Steuerung des Kassettenrekorders sind bei einer Umbesetzung der Speicheradressierung unbedingt zu beachten, so daß also eine Umschaltung von Basic-ROM auf RAM durch POKE 1,54 erledigt werden kann.

Schaltet man durch den ebengenannten POKE-Befehl das Basic aus, so ist man jedoch voll und ganz auf seine Künste als Maschinenspracheprogrammierer angewiesen. Es sei denn, man überträgt die Daten des Basic-ROM's in das darunterliegende RAM. Dies ist z.B. sinnvoll, wenn man nur Teile des Basic's benutzen will — nur für Profis zu empfehlen — oder schlicht und einfach den Basic-Interpreter ändern will.

Wahrscheinlich werden Sie das Auslesen des Wertes aus einer Speicherzelle und das Einschreiben in die gleiche Speicherzelle für ziemlich sinnlos halten, da sich ja doch nichts ändert. Aber wir hatten eben bereits erwähnt, daß bei der Einschaltbelegung im Byte 1 des RAM's ein POKE-Befehl auf das RAM adressiert wird und einem PEEK-Befehl das ROM zugrundegelegt ist. Durch folgende kleine Programmschleife können Sie den Inhalt des Basic-ROM's in das darunterliegende RAM kopieren:

```
FOR I=40960 TO 49151
  POKE I, PEEK (I)
NEXT
```

Nach Ausführung dieses kleinen Programmes können Sie mit POKE 1,54 aufs RAM umschalten, was dem Anwender nach außenhin natürlich nicht in Erscheinung tritt. Eine Änderung des Basic-RAM's wollen wir an dieser Stelle jedoch nicht weiter verfolgen.

Wollen Sie das Betriebssystem auch umschalten — wie wir es z.B. später beim Ausladen unseres Assemblers verwenden werden — so können Sie durch die vorgenannte FOR...NEXT-Schleife den Adreßbereich von 57344 bis 65535 ins RAM übertragen, und anschließend mit POKE 1,53 (POKE 1,52, wenn auch das Basic ausgeblendet werden soll) übertragen.

Letztere Methode wird z.B. verwendet, um den Grafikspeicher aus dem Basic-RAM herauszuziehen und unters Kernal zu legen. Dadurch kann auch das komplette Basic-RAM bei Grafikanwendungen für das Programm benutzt werden.

Wer seinen Zeichensatz ändern möchte, ist darauf angewiesen, daß er auch den Zeichengenerator byteweise auslesen kann, wodurch sich aus den Bitmustern die einzelnen Zeichen zusammensetzen. Mit POKE 1,51 kann man mit dem PEEK-Befehl auf das Zeichengenerator-ROM zugreifen, die Zeichen in einen freien RAM-Bereich holen und dort den eigenen Wünschen anpassen.

Auf das Thema Motorsteuerung der Datasette werden wir bei Adresse 192 noch etwas genauer eingehen.

3/1.2.2

Page 1 bis Page 3

Ebenso wie bei der Zero-Page wollen wir auch bei den weiteren Adressen bis zur Speicherzelle 1023 zunächst eine Übersicht bringen und anschließend jede Adresse einzeln ausführlich besprechen.

3/1.2.2.1

Übersicht

Hexadez.	Dezimal	Belegung
00FF-010A	255-266	Puffer für Umwandlung Fließkomma nach ASCII
0100-013E	256-318	Speicher für Korrektur bei Bandeingabe
0100-01FF	256-511	Prozessor Stack
0200-0258	512-600	BASIC-Eingabepuffer
0259-0262	601-610	Tabelle der logischen Dateinummern
0263-026C	611-620	Tabelle der Gerätenummern
026D-0276	621-630	Tabelle der Sekundäradresse
0277-0280	631-640	Tastaturpuffer
0281-0282	641-642	Start des BASIC-RAM
0283-0284	643-644	Ende des BASIC-RAM
0285	645	Timeout-Merker für IEC-Bus
0286	646	aktuelle Farbe
0287	647	Farbe unter dem Cursor
0288	648	High-Byte Video-RAM
0289	649	Länge des Tastaturpuffers
028A	650	Merker für Repeatfunktion aller Tasten
028B	651	Zähler für Repeatgeschwindigkeit
028C	652	Zähler für Repeatverzögerung
028D	653	Merker für Shift, Commodore und CTRL (Bit 0, 1 und 2)
028E	654	wie $028D
028F-0290	655-656	Vektor für Tastatur-Dekodierung
0291	657	Merker für SHIFT/Commodore gesperrt
0292	658	Merker für Scrollen

Hexadez.	Dezimal	Belegung
0293	659	RS 232 Kontrollwort
0294	660	RS 232 Befehlswort
0295-0296	661-662	Bit-Timing
0297	663	RS 232 Status
0298-029A	664-666	RS 232 Baud-Rate
029B	667	Zeiger auf empfangenes Byte RS 232
029C	668	Zeiger auf Input von RS 232
029D	669	Zeiger auf zu übertragendes Byte Rs 232
029E	670	Zeiger auf Ausgabe RS 232
029F-02A0	671-672	Speicher für IRQ während Bandbetrieb
02A1	673	CIA 2 NMI-Merker
02A2	674	CIA 1 Timer A
02A3	675	CIA 1 Interrupt-Merker
02A4	676	CIA 1 Merker für Timer A
02A5	677	aktuelle Bildschirmzeile
02A6	678	Merker für PAL- (1) oder NTSC-Version (0)
02C0-02FE	704- 766	Platz für Sprite (Block 11)
0300-0301	768- 769	$E38B Vektor für BASIC-Warmstart
0302-0303	770- 771	$A483 Vektor für Eingabe einer Zeile
0304-0305	772- 773	$A57C Vektor für Umwandlung in Interpretercode
0306-0307	774- 775	$A71A Vektor für Umwandlung in Belegung Klartext (LIST)
0308-0309	776- 777	$A7E4 Vektor für BASIC-Befehlsadresse holen
030A-030B	778- 779	$AEB6 Vektor für Ausdruck auswerten
030C	780	Akku für SYS-Befehl
030D	781	X-Register für SYS-Befehl
030E	782	Y-Register für SYS-Befehl
030F	783	Status-Register für SYS-Befehl
03010	784	$4C JMP-Befehl für USR-Funktion
0311-0312	785- 786	USR-Vektor
0314-0315	788- 789	IRQ-Vektor
0316-0317	790- 791	BRK-Vektor
0318-0319	792- 793	NMI-Vektor
031A-031B	794- 795	OPEN-Vektor
031C-031D	796- 797	CLOSE-Vektor
031E-031F	798- 799	CHKIN-Vektor
0320-0321	800- 801	CKOUT-Vektor
0322-0323	803- 803	CLRCH-Vektor
0324-0325	804- 805	INPUT-Vektor
0326-0327	806- 807	OUTPUT-Vektor
0328-0329	808- 809	STOP-Vektor
032A-032B	810- 811	GET-Vektor
032C-032D	812- 813	CLALL-Vektor
032E-032F	814- 815	Warmstart-Vektor
0330-0331	816- 817	LOAD-Vektor
0322-0333	818- 819	SAVE-Vektor
033C-03FB	828-1019	Bandpuffer
0340-037E	832- 894	Platz für Sprite (Block 13)
0380-03BE	896- 958	Platz für Sprite (Block 14)
03C0-03FE	960-1022	Platz für Sprite (Block 15)

3/1.2.3

VIC-Register

Mit '*' bezeichnete Register sind für alle Sprites zuständig: je Sprite ein Bit, korrespondierend in den Nummern (Sprite 0 — Bit 0, . . ., Sprite 7 — Bit 7).

Register	Adresse	Bedeutung
0	53248	Basisadresse V
0	53248	Sprite 0: X-Position
1	53249	Sprite 0: Y-Position
2	53250	Sprite 1: X-Position
3	53251	Sprite 1: Y-Position
4	53252	Sprite 2: X-Position
5	53253	Sprite 2: Y-Position
6	53254	Sprite 3: X-Position
7	53255	Sprite 3: Y-Position
8	53256	Sprite 4: X-Position
9	53257	Sprite 4: Y-Position
10	53258	Sprite 5: X-Position
11	53259	Sprite 5: Y-Position
12	53260	Sprite 6: X-Position
13	53261	Sprite 6: Y-Position
14	53262	Sprite 7: X-Position
15	53263	Sprite 7: Y-Position
16	53264	*Sprite 0-7: MSB der X-Position
17	53265	Steuerregister 1 Bits 0-2: Weiches Abrollen (Y-Richtung) Bit 3: 24/25 Zeilenwahl (24 Zeilen: 0) Bit 4: Bildschirm ausschalten (=0) Bit 5: Bit-Map-Modus einschalten (=1) Bit 6: Erweiterter Hintergrundmodus (=1) Bit 7: Rasterwertregister (MSB)
18	53266	Rasterwertregister (Bits 0-7)
19	53267	Lichtgriffel (X-Richtung)
20	53268	Lichtgriffel (Y-Richtung)
21	53269	*Sprites ein/aus (1=ein)
22	53270	Steuerregister 2 Bits 0-2: Weiches Abrollen (X-Richtung) Bit 3: 39/40 Spaltenwahl (40 Spalten: 1) Bit 4: Vielfarbenmodus (=1) Bits 5-7: nicht genutzt
23	53271	Sprite-Vergrößerungsregister (X-Richtung)

Register	Adresse	Bedeutung
24	53272	Adreß-Kontrollregister Bit 0: nicht benutzt Bits 1-3: Adresse des Zeichengenerators bzw. Bit-Map Bits 4-7: Adresse des Bildschirmspeichers
25	53273	Unterbrechungs-Flaggenregister Bit 0: Rastervergleich Bit 1: Zusammenstoß Sprite — Hintergrund Bit 2: Zusammenstoß Sprite — Sprite Bit 3: Unterbrechung durch Lichtgriffel Bits 4-6: nicht benutzt Bit 7: eines der Bits 0-3 gesetzt
26	53274	Unterbrechungs-Maskenregister Bit 0: Rastervergleich (1 = ein) Bit 1: Zusammenstoß Sprite — Hintergrund Bit 2: Zusammenstoß Sprite — Sprite Bit 3: Unterbrechung durch Lichtgriffel Bits 4-7: nicht benutzt
27	53275	*Prioritätsregister Sprite — Hintergrund (1 = Sprite)
28	53276	*Sprite-Multi-Color-Steuerregister (1 = Multicolormodus)
29	53277	*Sprite-Vergrößerungsregister (Y-Richtung)
30	53278	*Kollisionsregister Sprite — Sprite
31	53279	*Kollisionsregister Sprite — Hintergrund
32	53280	Rahmenfarbe
33	53281	Hintergrundfarbe 1
34	53282	Hintergrundfarbe 2
35	53283	Hintergrundfarbe 2
36	53284	Hintergrundfarbe 3
37	53285	Sprite-Vielfarbenregister 0
38	53286	Sprite-Vielfarbenregister 1
39	53287	Sprite 0: Farbregister
40	53288	Sprite 1: Farbregister
41	53289	Sprite 2: Farbregister
42	53290	Sprite 3: Farbregister
43	53291	Sprite 4: Farbregister
44	53292	Sprite 5: Farbregister
45	53293	Sprite 6: Farbregister
46	53294	Sprite 7: Farbregister

3/1.2.4

SID-Register

Register	Adresse	Bedeutung
0	**54272**	**Basisadresse SI**
0-24	54272-54296	SID-Chip Steuerregister (können nur beschrieben werden)
0-6	**54272-54278**	**Stimme 1**
0	54272	Frequenz (Low Byte)
1	54273	Frequenz (High Byte)
2	54274	Impulsbreite (Low Byte)
3	54275	Impulsbreite (High Byte)
4	54276	Klangkontrollregister Bit 0: Key-Bit (Ton ein/aus) Bit 1: Synchronisation mit Stimme 3 Bit 2: Ringmodulation mit Stimme 3 Bit 3: Testbit (normal unbenutzt) Bit 4: Dreieckswelle Bit 5: Sägezahnwelle Bit 6: Rechteckwelle Bit 7: Rauschen
5	54277	Steuerregister für Einsatz und Ausklingen Bits 0-3: Wert für Ausklingen (Decay) Bits 4-7: Wert für Toneinsatz (Attack)
6	54278	Steuerregister für Halten und Nachklingen Bits 0-3: Wert für Nachklingen (Release) Bits 4-7: Wert für Halten (Sustain)
7-13	**54279-54285**	**Stimme 2**
7	54279	Frequenz (Low Byte)
8	54280	Frequenz (High Byte)
9	54281	Impulsbreite (Low Byte)
10	54282	Impulsbreite (High Byte)
11	54283	Klangkontrollregister Bit 0: Key-Bit (Ton ein/aus) Bit 1: Synchronisation mit Stimme 1 Bit 2: Ringmodulation mit Stimme 1 Bit 3: Testbit (normal unbenutzt) Bit 4: Dreieckswelle Bit 5: Sägezahnwelle Bit 6: Rechteckwelle Bit 7: Rauschen

Register	Adresse	Bedeutung
12	54284	Steuerregister für Einsatz und Ausklingen Bits 0-3: Wert für Ausklingen (Decay) Bits 4-7: Wert für Toneinsatz (Attack)
13	54285 SI+13	Steuerregister für Halten und Nachklingen Bits 0-3: Wert für Nachklingen (Release) Bits 4-7: Wert für Halten (Sustain)
14-20	**54286-54292**	**Stimme 3**
14	54286	Frequenz (Low Byte)
15	54287	Frequenz (High Byte)
16	54288	Impulsbreite (Low Byte)
17	54289	Impulsbreite (High Byte)
18	54290	Klangkontrollregister Bit 0: Key-Bit (Ton ein/aus) Bit 1: Synchronisation mit Stimme 2 Bit 2: Ringmodulation mit Stimme 2 Bit 3: Testbit (normal unbenutzt) Bit 4: Dreieckswelle Bit 5: Sägezahnwelle Bit 6: Rechteckwelle Bit 7: Rauschen
19	54291	Steuerregister für Einsatz und Ausklingen Bits 0-3: Wert für Ausklingen (Decay) Bits 4-7: Wert für Toneinsatz (Attack)
20	54292	Steuerregister für Halten und Nachklingen Bits 0-3: Wert für Nachklingen (Release) Bits 4-7: Wert für Halten (Sustain)
21-24	**54293-54296**	**Klangfilterfunktionen**
21	54293	Cutoff-Frequenz des Filters (Low Byte)
22	54294	Cutoff-Frequenz des Filters (High Byte)
23	54295	Filter/Resonanz-Kontrollregister (1 = ein) Bit 0: Filter für Stimme 1 Bit 1: Filter für Stimme 2 Bit 2: Filter für Stimme 3 Bit 3: Filter für externes Signal Bits 4-7: Resonanzwert
24	54296	Modus- und Lautstärke Bits 0-3: Lautstärkenkontrollregister Bit 4: Tiefpaßfilter Bit 5: Bandpaßfilter Bit 7: Stimme 3 ein/aus (1 = aus)
Register, die nur gelesen werden können		
25	54297	Paddle X-Wert
26	54298	Paddle Y-Wert
27	54299	Momentaner Digitalwert von Stimme 3
28	54300	Hüllkurvenregister (Stimme 3)

3/1.2.5

CIA-Register
(Complex Interface Adapter)

		56320-56335 CIA 1
Register	**Bit 76543210**	**Bedeutung**
56320		Datenregister Port A
		Tastatur:
	XXXXXXXX	Spaltenansteuerung der Matrix
		Anschlüsse an Control-Port 2
	 XXXX	Richtung (Joystick)
	. . . X	Feuerknopf (Joystick)
	. . . . XX . .	Paddle-Feuerknöpfe
		Analogschalter:
	XX	Paddlesatzauswahl Control Port 1 oder 2
56321		Datenregister Port B
		Tastatur:
	XXXXXXXX	Zeilenabfrage der Matrix
		Anschlüsse an Control-Port 1:
	 XXXX	Richtung (Joystick)
	. . . X	Feuerknopf (Joystick)
	. . . . XX . .	Paddle-Feuerknöpfe
56322		Datenrichtungsregister Port A
56323		Datenrichtungsregister Port B
56324-56327		Timer
56324		Timer A (Low Byte) (für IRQ-Frequenz)
56325		Timer A (High Byte) (für IRQ-Frequenz)
56326		Timer A (Low Byte)
56327		Timer B (High Byte)
56238-56331		Realzeituhr
56328		Zehntelsekunden
56329		Sekunden (BCD)
56330		Minuten (BCD)
56331	. XXXXXXX	Stunden (BCD)
	X	AM/PM (Anzeige vormittag/nachmittag)

Register	Bit 76543210	Bedeutung
		56320-56335 CIA 1
56332		Serieller E/A-Puffer (synchron)
56333		Unterbrechungs-Kontrollregister
56334		Steuerregister A
56335		Steuerregister B
		56576-56831 CIA 2
56576		Datenregister Port A
	 XX	Segmentwahl des VIC-II-Chip 00 Segment 3 (49152 - 65535) 01 Segment 2 (32768 - 49151) 10 Segment 1 (16384 - 32767) 11 Segment 0 (00000 - 16383)
	. . . . . X . .	RS 232 Senderdaten-Ausgabe (TXD)
	. . . . X . . .	ATN-Ausgangssignal
	. . . X	Serieller Bus, Ausgabe Takt
	. . X	Serieller Bus, Ausgabe Daten
	. X	Serieller Bus, Eingabe Takt
	X	Serieller Bus, Eingabe Daten
56577		Datenregister Port B
	 X	RS 232 Daten empfangen (RXD)
	. . . . . . X .	RS 232 Sendeaufforderung (RTS)
	. . . . . X . .	RS 232 Datenterminal bereit (DTR)
	. . . . X . . .	RS 232 Ringindikator (RI)
	. . . X	RS 232 Träger entdeckt (DCD)
	. X	RS 232 Bereit zum Senden (CTS)
	X	Datensatz bereit (DSR)
56378		Datenrichtungsregister Port A
56379		Datenrichtungsregister Port B
56380-56383		Timer
56380		Timer A (Low Byte)
56381		Timer A (High Byte)
56382		Timer A (Low Byte)
56383		Timer B (High Byte)
56384-56387		Realzeituhr
56384		Zehntelsekunden
56385		Sekunden
56386		Minuten
56387	. XXXXXXX	Stunden
	X	AM/PM (Anzeige vormittag/nachmittag)
56388		Serieller E/A-Puffer (synchron)
56389		Unterbrechungs-Kontrollregister
56390		Steuerregister A
56391		Steuerregister B

3/2
Der Basic-Interpreter des C 64

In diesem Kapitel wird der Basic-Interpreter des C 64 genauer betrachtet. Wir werden zunächst seinen generellen Aufbau besprechen und die Adressen der wichtigsten Routinen im Basic-ROM zusammenstellen. Danach folgt dann ein komplettes, disassembliertes ROM-Listing, welches Ihnen ermöglicht, die Routinen für Ihre Programme effizient zu nutzen.

3/2.1
Übersicht über den Basic Interpreter

Der Basic-Interpreter des C 64 beinhaltet alle Routinen, die zum Schreiben und Bearbeiten von Basic-Programmen benötigt werden. Dies sind im wesentlichen Routinen zur Eingabe von Programmzeilen, zur Ausführung von Befehlen sowie für arithmetische Operationen.

Die Routinen zur Eingabe von Programmzeilen liegen am Anfang des Interpreters. Sie bestehen aus einer Routine zum Löschen von Zeilen sowie einer Routine zum Einfügen von Programmzeilen. Soll eine Zeile nur verändert werden, so muß diese erst gelöscht und dann wieder eingefügt werden. Zu der Zeileneingabe gehören außerdem Routinen, die den eingegebenen Text in eine interne Darstellung wandeln, bei der jedes Basic-Schlüsselwort wie z.B. RUN, GOTO, +, SIN etc. in ein Token (ein 1-Byte-Wert mit gesetzten Bit 7) umgewandelt wird. Um diese Darstellung wieder ausgeben zu können, existiert natürlich auch eine Routine, die eine Basic-Zeile in Klartext ausgibt. Diese wird vom LIST-Befehl genutzt. Diese Routinen können genutzt werden, um z.B. ein kleines Textverarbeitungssystem zu schreiben, oder den Editor um einige Eingabehilfen zu erweitern.

Nach dem Basic-Start befindet sich der Basic-Interpreter in der Eingabewarteschleife. Hier wird solange verweilt, bis von der Tastatur eine Zeile eingegeben und mit <RETURN> abgeschlossen wurde. Die eingegebene Zeile wird dann in die interne Darstellung gewandelt. Beginnt die Zeile mit einer Zeilennummer, so wird eine evtl. vorhandene Zeile mit der gleichen Zeilennummer gelöscht. Dann wird die neue Zeile in den vorhandenen Text eingefügt. Ist keine Zeilennummer vorhanden, so handelt es sich um Befehle, die sofort ausgeführt werden sollen. Dies ist der sogenannte Direktmodus. Die Routinen der Befehle werden dann in der Interpreterschleife nacheinander aufgerufen.

Wichtig für den Assemblerprogrammierer sind außerdem die arithmetischen Operationen, die der C 64 bietet. Wer in Assemblerprogrammen mit Fließkommazahlen arbeiten will, kann sich viel Arbeit ersparen, indem er die Routinen des Basic-Interpreters benutzt. Alle Fließkommafunktionen werden im C 64 zwischen zwei Speicherbereichen, in denen die Fließkommazahlen stehen müssen, durchgeführt. Diese Bereiche werden mit FAC und ARG bezeichnet. Im Basic-Interpreter finden sich Routinen zur Berechnung der folgenden Funktionen: +, —, *, /, ↑, SGN, INT, ABS, SQR, LOG, EXP, COS, SIN, TAN, und ATN.

Wenn Sie den Basic-Interpreter verändern oder erweitern wollen, sollten Sie noch wissen, wie Programm und Daten in den Speicherbereich abgelegt werden. Basic nutzt den ihm zugewiesenen Bereich (Standard: $801-$9FFF) voll aus. Am Anfang des Bereichs wird das Programm abgelegt. Direkt im Anschluß an das Programm folgen die normalen Variablen, die während des Programmlaufs angelegt werden. Die Startadresse dieses Bereichs steht in $2D/2E. Daran schließen sich die Arrays an. Die Startadresse dieses Blocks finden Sie in $2F/$30, und die Endadresse (+1) in $31/$32. Bei Stringvariablen werden nur der Name, die Länge, sowie die Adresse der Strings in der Tabelle abgelegt. Ist der String im Programm konstant angegeben, wird diese Adresse in die Tabelle übernommen. Bei einem eingegebenen oder berechneten String, wird dieser am Ende des Speicherbereichs abgelegt und seine Adresse in die Tabelle übernommen. Die folgende Skizze soll den Speicheraufbau des C 64 verdeutlichen:

Startadresse des Bereichs	Speicherinhalt	
($2B/$2C) →	Programmtext	← (43,44)
($2D/$2E) →	normale Variablen	← (45,46)
($2F/$30) →	indizierte Variablen (Arrays)	← (47,48)
($31/$32) →	! v ∧ !	← (49,50)
($33/$34) →	Zeichenketten (Strings)	← (51,52)
($37/$38) →		← (55,56)

3/2.2

Die wichtigsten Adressen des Basic-Interpreters

In der folgenden Tabelle finden Sie eine nahezu vollständige Auflistung der Routinen des Basic-Interpreters, die Sie für eigene Anwendungen benutzen können. Die Liste ist nach Adressen geordnet. Teilweise sind die Ein- und Ausgabeparameter der Routinen mit angesprochen. Sie sollten sich daher die Routinen im ROM-Listing des Basic-Interpreters (Kapitel 3/2.3) genau ansehen.

Adr.	Name	Funktion der Routine
$A38A	STAPSUCH	Stapelsuchroutine für FOR und GOSUB
$A3B8	BLOCKMOV	Blockverschieberoutine zu höheren Adressen. Eingabe: Alter Blockanfang in $5F/$60, altes in $5A/$5B, neues Blockende in $58/$59
$A3FB	STAPVOLL	Prüft, ob Akku*2 Byte im Stapel frei sind
$A408	MAKESPC	Schafft Platz im Speicher bis zur Adresse in A/Y
$A435	ERROUTM	Ausgabe 'out of memory'
$A437	ERROR	Fehlermeldung mit Nummer in X ausgeben
$A474	READY	Ausgabe 'ready.'
$A480	INLOOP	Eingabewarteschleife
$A49C	PRGZLE	Löschen & Einfügen von Programmzeilen
$A4A9	PRGZLOE	Löschen einer Programmzeile
$A4ED	PRGZEINF	Einfügen einer Programmzeile
$A533	PRGZBIND	BASIC-Programmzeilen neu binden
$A560	GETZEI	Eine Zeile in Puffer ab $200 einlesen
$A571	ERRSTRRTL	Ausgabe 'string too long'
$A579	PRGZUM	Umwandlung einer Programmzeile in die interne Darstellung
$A613	GETPZADR	Startadresse der Basic Zeile mit Nummer in $14/$15 in $5F/$60 liefern. Bei einem Fehler wird Carry=0
$A642	BBNEW	NEW-Befehl
$A65E	BBCLR	CLR-Befehl
$A68E	PRGBEGIN	Programmzeiger auf Basic-Start setzen
$A69C	BBLIST	LIST-Befehl
$A717	KLARTEXT	interne Darstellung in Text wandeln und ausgeben

2.2 Die wichtigsten Adressen des Basic-Interpreters

Adr.	Name	Funktion der Routine
$A742	BBFOR	FOR-Befehl
$A7AE	INTLOOP	Interpreterschleife
$A7ED	EXETOKEN	Führt Basic-Befehl mit Token im Akku aus
$A81D	BBRESTOR	RESTORE-Befehl
$A82C	CHSTOP	prüft die STOP-Taste und bricht ggf. das Programm ab
$A82F	BBSTOP	STOP-Befehl
$A831	BBEND	END-Befehl
$A857	BBCONT	CONT-Befehl
$A871	BBRUN	RUN-Befehl
$A883	BBGOSUB	GOSUB-Befehl
$A8A0	BBGOTO	GOTO-Befehl
$A8D2	BBRETURN	RETURN-Befehl
$A8F8	BBDATA	DATA-Befehl
$A906	NEXTST	suche nächste Anweisung im Programm
$A909	NEXTLINE	suchen nächste Programmzeile
$A928	BBIF	IF-Befehl
$A93B	BBREM	REM-Befehl
$A94B	BBON	ON-Befehl
$A96B	BASZADR	holt Zeilennummer in $14/$15
$A9A5	BBLET	LET-Befehl
$A9BA	WERTZUW	Wertzuweisung an Variable mit Adresse in $49/$4A, Typ-Flag für String/num. im Akku und Integer-Flag im Stack
$A9C4	WERTZUWI	Integerzuweisung an Var. mit Adr. $49/$4A
$A9D6	WERTZUWR	Realzuweisung an Var. mit Adr. $49/$4A
$AA2C	WERTZUWS	Stringzuweisung an Var. mit Adr. $49/$4A Adresse des Stringdeskriptors in $64/$65
$AA80	BBPRIDEV	PRINT#-Befehl
$AA86	BBCMD	CMD-Befehl
$AAA0	BBPRINT	PRINT-Befehl
$AB1E	STROUT	String ausgeben
$AB3B	SPCOUT	Ein Leerzeichen ausgeben; Bei $13=0 Cursorright ausgeben
$AB3F	SPOUT	Ein Leerzeichen ausgeben
$AB42	CURIOUT	Cursor right ausgeben
$AB45	FRAGOUT	Fragezeichen ausgeben
$AB7B	BBGET	GET-Befehl
$ABA5	BBINPDEV	INPUT#-Befehl
$ABA0	BCLRCH	setzt die Ein- und Ausgabe auf Tastatur bzw. Bildschirm
$ABBF	BBINPUT	INPUT-Befehl
$AC06	BBREAD	READ-Befehl
$AD1D	BBNEXT	NEXT-Befehl
$AD8A	FRMNUM	holt Ausdruck und prüft auf numerisch
$AD8D	CHKNUM	prüft auf numerisch
$AD8F	CHKSTR	prüft auf String
$AD99	ERRTYPM	Ausgabe 'type mismatch'

2.2 Die wichtigsten Adressen des Basic-Interpreters Teil 3: Interne Software

Adr.	Name	Funktion der Routine
$AD9E	FRMEVL	holt einen Ausdruck. Bei numerisch steht der Wert im FAC, bei String kann er mit FRESTR geholt werden.
$AE83	GETARI	arithm. Ausdruck holen
$AEA8	PI	KONSTANTE PI (3.14) im Fließkommaformat
$AED4	BBNOT	NOT-Befehl
$AEE3	FKT	User-Funktion ausführen
$AEEA	BBSGN	SGN-Funktion
$AEF1	FRMKLA	holt Ausdruck in Klammern
$AEF7	CHKKLAZU	prüft, ob ')' folgt
$AEFA	CHKKLAUF	prüft, ob '(' folgt
$AEFD	CHKKOM	prüft, ob ',' folgt
$AEFF	CHKZEI	prüft, ob Zeichen im Akku folgt
$AF08	ERRSYNT	Ausgabe 'syntax error'
$AF28	VARWERT	holt Wert von Variable aus Basic-Text
$AF84	GETTIME	Zeit holen
$AFE6	BBOR	OR-Befehl
$AFE9	BBAND	AND-Befehl
$B016	VERGL	Vergleich ARG mit FAC. Ergebnis im FAC
$B081	BBDIM	DIM-Befehl
$B08B	VARSUCH	Variableadresse in $47/$48 liefern
$B0E7	NAMSUCH	liefert Adresse der Variablen mit Namen aus $45/$46 in Adresse $47/$48
$B113	CHLETT	prüft auf Buchstabe. Ja-> Carry=1
$B194	ARRBEG	liefert Zeiger auf erstes Arrayelement
$B1A5		KONSTANTE -32768 im Fließkommaformat
$B1AA	FLPAY	Wandelt FAC in Integer in Akku/Y
$B245	ERRBADS	Ausgabe 'bad subscipt'
$B248	ILLQERR	Ausgabe 'illegal quantity'
$B34C	ARRSIZ	Ermittelt die Arraygröße
$B37D	BBFRE	FRE-Befehl
$B39E	BBPOS	POS-Befehl
$B3A6	CHKDIR	testet auf Direktmodus. Direkmodus führt zu 'illegal direct'
$B3AB	ERRILLD	Ausgabe 'illegal direct'
$B3AE	ERRUNFN	Ausgabe 'undef'd function'
$B3B3	BBDEF	DEF-Befehl
$B3E1	CHFNSYN	prüft die FN-Syntax
$B3F4	BBFN	FN-Befehl
$B465	BBSTR	STR$-Befehl
$B475	STRZEIG	berechnet Zeiger auf String
$B487	NEUSTR	richtet einen neuen String ein
$B526	GARCOL	Garbage Collection, räumt den Stringbereich auf
$B63D	STRADD	Stringverknüpfung '+'
$B643	FRESTR	liefert String
$B6EC	BBCHR	CHR$-Befehl
$B700	BBLEFT	LEFT$-Befehl

Adr.	Name	Funktion der Routine
$B72C	BBRIGHT	RIGHT$-Befehl
$B737	BBMID	MID$-Befehl
$B77C	BBLEN	LEN-Befehl
$B782	STRPAR	Stringparameter holen
$B78B	BBASC	ASC-Befehl
$B79B	GETBYTE	CHKCOM und Wert 0-255 aus Basictext holen
$B7EB	GETAB	Aufrufe: CHKCOM, FRMNUM, FACADR, GETBYTE
$B7F7	FACADR	Wandelt FAC in Integer in $14/$15
$B7AD	BBVAL	VAL-Befehl
$B80D	BBPEEK	PEEK-Befehl
$B824	BBPOKE	POKE-Befehl
$B82D	BBWAIT	WAIT-Befehl
$B849	ADD05	FAC:=FAC+0.5
$B850	FKMINUS	FAC:=Konstante(A/Y)-FAC
$B853	FAMINUS	FAC:=ARG-FAC
$B867	FKPLUS	FAC:=Konstante(A/Y)+FAC
$B86A	FAPLUS	FAC:=FAC+ARG
$B9EA	BBLOG	LOG-Befehl
$B97E	ERROVFL	Ausgabe 'overflow'
$B9BC	FKLOG	Konstanten für LOG
$BA28	FKMAL	FAC:=Konstante(A/Y)*FAC
$BA2B	FAMAL	FAC:=ARG*FAC
$BA8C	LDARG	ARG:=Konstante(A/Y)
$BAE2	FMAL10	FAC:=FAC*10
$BAF9		KONSTANTE 10
$BAFE	FDURCH10	FAC:=FAC/10
$BB0F	FKDIV	FAC:=Konstante(A/Y)/FAC
$BB12	FADIV	FAC:=ARG/FAC
$BB8A	DIVZERR	Ausgabe 'division by zero'
$BBA2	LDFAC	FAC:=Konstante(A/Y)
$BBC7	FACA4	AKKU#4:=FAC
$BBCA	FACA3	AKKU#3:=FAC
$BBD0	FACVAR	Variable:=FAC Variablenadr. in $49/$4A
$BBFC	ARGFAC	FAC:=ARG
$BC0C	FACARG	ARG:=FAC
$BC1B	FACRUND	FAC runden
$BC2B	FSGN	Vorzeeichen von FAC holen
$BC39	BBSGN	SGN-Befehl
$BC58	BBABS	ABS-Befehl
$BC5B	FVGLAY	Konstante(A/Y) mit FAC vergleichen
$BCCC	BBINT	INT-Befehl
$BCF3	ASCFLP	Umwandlung von ASCII (Text) in Fließkomma
$BDB3		KONST.: 99999999.9 999999999 1000000000
$BDC2	ERRZNR	Ausgabe 'in ' & Zeilennummer bei ERROR
$BDCD	ADROUT	Zahl A/X dezimal ausgeben
$BDDD	FACASC	FAC nach ASCII (Text ab $100) wandeln
$BF71	BBSQR	SQR-Befehl
$BF78	FKHOCH	FAC:=ARG^Konstante(A/Y)

Adr.	Name	Funktion der Routine
$BF7B	FAHOCH	FAC:=ARG^FAC
$BFBF		Konstanten für EXP
$BFE8		KONSTANTE: 1
$BFED	BBEXP	EXP-Befehl
$E043	POLY1	Polynomberechnung: $a_1x+a_2x^3+a_3x\ +a_4x^5+...$
$E059	POLY0	Polynomberechnung: $a_0+a_1x+a_2x^2+a_3x^3+...$
$E08D		Konstanten für RND
$E097	BBRND	RND-Befehl
$E107	OUTBREAK	Ausgabe 'break'
$E10C	BBSOUT	Ausgabe eines Zeichens
$E112	BBASIN	Einlesen eines Zeichens
$E118	BCHKOUT	Ausgabegerät festlegen
$E11E	BCHKIN	Eingabegerät festlegen
$E124	BGETIN	Ein Zeichen per GET holen
$E12A	BBSYS	SYS-Befehl
$E156	BBSAVE	SAVE-Befehl
$E165	BBVERIFY	VERIFY-Befehl
$E168	BBLOAD	LOAD-Befehl
$E1BE	BBOPEN	OPEN-Befehl
$E1C7	BBCLOSE	CLOSE-Befehl
$E1D4	PARLOSA	Parameter für LOAD und SAVE holen
$E219	PAROPEN	Parameter für OPEN holen
$E264	BBCOS	COS-Befehl
$E26B	BBSIN	SIN-Befehl
$E2B4	BBTAN	TAN-Befehl
$E2E0		Konstanten für SIN und COS
$E30E	BBATN	ATN-Befehl
$E33E		Konstanten für ATN
$E37B	BNMI	Basic-NMI Routine
$E394	BKALT	Basic-Kaltstart
$E3A2	KCHRGET	Kopie der CHRGET-Routine
$E3BF	BINIT	Basic-Initialisierungsroutine
$E447	BVECTAB	Tabelle des Basic-Standardvektoren
$E453	BVECLAD	Laden der Vektoren
$E45F	BMELD	Text der Einschaltmeldung
$E4E0	WARTE	Warten max. 8.5 Sekunden auf die C= Taste

Teil 4

Software-Erstellung

4/2

Basic 2.0

4/2.1

Grundlagen

Bevor man einen Rechner programmieren kann, muß man sich zunächst mit seinen Eigenschaften vertraut machen. Insbesondere die Handhabung sollte während einer Programmierphase nicht noch zusätzliche Schwierigkeiten aufwerfen. Auch sollte man sich mit den Editiermöglichkeiten (Erfassen eines Programmes) vertraut machen.

Das vorliegende Kapitel beschäftigt sich zunächst mit den drei Betriebsarten des C 64 und geht im nächsten Abschnitt auf eine dieser Betriebsarten näher ein.

Eine grundlegende Eigenschaft von Programmen sind Variablen, in denen Zahlen oder Zeichenreihen für Berechnungen vorgehalten werden können. Kapitel 4/2.1.3 erklärt Grundlegendes über Variablen.

4/2.1.1

Betriebsarten

Ihr Rechner kennt drei sogenannte Betriebsarten. Die Betriebsart, in der Sie sich nach dem Einschalten befinden, wird Direktmodus genannt, da Eingaben an den Rechner direkt von ihm ausgeführt werden.

Der nächste Modus wird Editiermodus genannt. In gewissem Sinne ist er auch ein Direktmodus, jedoch werden hier keine allgemeinen Arbeiten erledigt, sondern die Editierung eines Programmes vorgenommen; daher der Name.

Als letztes ist der Programm- oder auch Rechenmodus zu nennen. Dieser Modus ist aktiv, wenn Sie ein Programm gestartet haben.

Im folgenden wollen wir nur auf den Direktmodus explizit eingehen, da sich die beiden anderen Modi von selbst ergeben werden. Wenn Sie länger mit dem Rechner

gearbeitet haben, werden Sie sich sicherlich keine Gedanken mehr darüber machen, in welchem Modus Sie arbeiten. Dies ist auch gar nicht nötig.

4/2.1.2

Direktmodus

Verbleiben wir noch einige Zeit in dem Modus, den wir nach Einschalten des Rechners vorliegen haben: im Direktmodus. Hier wollen wir erste kleine Rechenaufgaben erledigen.

Ihr Computer wird nicht umsonst als Rechner bezeichnet. Rechenaufgaben erfüllt er nicht nur in Form von Programmen, sondern auch im Direktmodus, ähnlich einem Taschenrechner. Fragen Sie Ihren Rechner doch mal nach dem Ergebnis der trivialsten aller Rechnungen, indem Sie ein Fragezeichen in Verbindung mit der Frage in folgender Form eingeben:

```
?1+1
```

Wenn Sie nun die RETURN-Taste drücken, erscheint unter Ihrer Frage direkt die Antwort, selbstverständlich eine „2". Darunter befindet sich eine Leerzeile, gefolgt von der obligatorischen „READY"-Meldung. Darunter blinkt wieder der Cursor und erwartet eine neue Eingabe von Ihnen.

Probieren Sie nun einzelne Rechnungen, indem sie jeder Rechnung ein Fragezeichen voranstellen. Als mathematische Operatoren bedienen sie sich des „+"-Zeichens für die Addition, des „-"-Zeichens für die Subtraktion, des „*" für die Multiplikation und „/" für die Division. Als Potenzierung dient der „Pfeil nach oben" links neben der RESTORE-Taste. Nicht zu verwechseln mit den Tasten für „Cursor nach oben".

Als Gegenstück für die Potenzierung wird das Wurzelziehen allerdings nicht durch eine Taste, sondern einen Befehl initiiert. Geben Sie z.B. ein

```
?SQR(2)
```

Wenn Sie anschließend die RETURN-Taste drücken, erscheint das Ergebnis in Form von 1.41421356 wieder unterhalb Ihrer Frage. In ähnlicher Weise sind auch andere

mathematische Funktionen zugelassen wie Sinus (SIN(Argument)), Cosinus (COS(Argument)) oder auch der Tangens (TAN(Argument)). Weitere mathematische Funktionen entnehmen Sie bitte dem Handbuch („Argumente“ gibt den jeweils umzurechnenden Wert an).

Merke: **Mit einem „?“ können im Direktmodus beliebige mathematische Formeln angegeben werden. Nach Drücken der RETURN-Taste ist das Ergebnis dieser Rechnung sofort auf dem Bildschirm sichtbar.**

Wenn der Rechner nur als besserer Taschenrechner dient, wäre er etwas zu teuer. Nähern wir uns seinen Möglichkeiten etwas weiter, indem wir Variablen verwenden.

4/2.1.3

Variablen

Aus der Algebra dürfte Ihnen bestimmt noch das „Rechnen“ mit Buchstaben geläufig sein. Ähnliches leistet auch Ihr Computer. Waren bei der Algebra die Buchstaben ein allgemeiner Platzhalter für irgendwelche Zahlen, so sind es allerdings in Ihrem Rechner konkrete Zahlen, die sich jedoch laufend verändern können.

Fangen wir mit einem kleinen Beispiel an. Geben Sie nacheinander in Ihren Rechner ein:

```
A=100 (RETURN-Taste)
B=123 (RETURN-Taste)
?A+B  (RETURN-Taste)
```

Es erscheint sofort das Ergebnis: 223. In diesem Falle hat die sogenannte Variable A zunächst den Wert 100 zugewiesen bekommen, die Variable B den Wert 123. Im Direktmodus haben wir dann gleich diese beiden Zahlen addiert, wobei im Gegensatz zur Algebra allerdings ein konkretes Ergebnis geliefert wurde.

Dieses kleine Beispiel hinkt zwar etwas, wie jedes Beispiel, da die Eingabe „?100+123“ sicherlich schneller vonstatten gegangen wäre. Der Sinn der Variablen wird aber gleich deutlich, wenn Sie eingeben

```
?A*B (RETURN-Taste)
```

Es erscheint sofort das Ergebnis 12300, ohne daß die Variablen erneut besetzt werden müssen. Auch die anderen mathematischen Operationen liefern sofort ein entsprechendes Ergebnis.

Nachdem wir nun die Variablen an einem kleinen Beispiel kennengelernt haben, wollen wir noch etwas Grundsätzliches zu dem Thema sagen. Variablen werden beim C 64 und 128 generell aus der Verbindung von ein oder zwei Buchstaben gebildet, wobei der zweite Buchstabe auch durch eine Ziffer von 1 bis 0 ersetzt werden kann. Eine Ziffer am Anfang eines sogenannten Variablennamens ist unzulässig. Auch Sonderzeichen sind nicht gestattet.

In unserem Beispiel haben wir die beiden Variablen A und B verwendet. Diese Variablen sind Platzhalter für Dezimalzahlen (Brüche als Kommazahlen) wie z.B. 0.25 oder 124.3333. Aber auch Variablen wie 100 und 123 (ganze Zahlen) können mit diesen Variablen dargestellt werden. Für ganze Zahlen (auch -1, -2, -3, . . .) gibt es noch einen gesonderten Variablentyp, als Integervariablen bezeichnet. Als Unterscheidungsmerkmal dient ein angehängtes „%"-Zeichen. Die Variablen A und A% können nebeneinander benutzt werden. A% kann allerdings keine Werte mit Nachkommastellen annehmen.

Ein weiterer Variablentyp sind die sogenannten String-Variablen oder Zeichenreihen-Variablen. In ihnen werden Buchstaben, aber auch Ziffern (ohne zahlmäßige Bedeutung), die üblichen Sonderzeichen, Grafikzeichen, aber auch Sonderzeichen in Form von Grafiksymbolen oder Steueranweisungen gespeichert. Den Zeichenreihen-Variablen sind gesonderte Befehle zugeordnet, die wir in Kapitel 4/2.4 besprechen werden.

Obwohl wir im weiteren Sprachgebrauch auch weiterhin die Bezeichnung Dezimalkomma verwenden wollen, sei an dieser Stelle darauf hingewiesen, daß Zahlen mit „Nachkommastellen" durch einen Punkt getrennt werden. Dies entspricht der amerikanischen Schreibweise. Also: nicht 1,25 für 1¼, sondern 1.25.

Merke: Dezimalzahlen werden mit einem Punkt dargestellt.

Nun noch etwas zu der sogenannten Lebensdauer von Variablen.

Eine Variable behält so lange ihren Wert, bis dieser geändert wird. Geben Sie jetzt z.B.

```
A=200 (RETURN-Taste)
```

ein, und anschließend

```
?A+B (RETURN-Taste)
```

so ist das Ergebnis 323, sofern Sie zwischendurch den Rechner nicht ausgeschaltet, und damit die Variablen gelöscht haben. Wir haben also den Wert in der Variablen A geändert. Ändern Sie nun auch den Wert in der Variablen B, indem Sie eingeben

```
B=400 (RETURN-TASTE)
```

und prüfen Sie das Ergebnis von

```
?A+B (RETURN-TASTE)
```

Die Werte der Variablen können Sie sich anschauen, wenn Sie lediglich den Variablennamen nach dem Fragezeichen eingeben. Geben Sie also ein

```
?A (RETURN-Taste)
?B (RETURN-Taste)
```

Neben den Neuzuweisungen eines Wertes an eine Variable gibt es noch einige andere Möglichkeiten, den Wert zu verändern, nämlich drei verschiedene Methoden, alle Variableninhalte zu löschen. Einerseits verlieren nach dem Aus- und Wiedereinschalten die Variablen ihren Wert (d.h. er ist 0 generell für alle Variablen nach dem Einschalten). Aber auch bei einem Programmstart mit RUN, wie wir später noch sehen werden, werden alle Werte von Variablen gelöscht. Die dritte Möglichkeit, alle Variablen zu löschen, beinhaltet der CLR-Befehl. Geben Sie also am Bildschirm ein

```
CLR (RETURN-Taste)
```

Wenn Sie nun eingeben

```
?A (RETURN-Taste)
?B (RETURN-Taste)
```

so sind die Werte in den Variablen null.

Merke: **Variablennamen werden aus zwei Buchstaben oder aus einem Buchstaben mit einer Ziffer gebildet.**

Der Wert einer Variablen nach dem Einschalten, nach einem Programmstart oder nach dem CLR-Befehl ist null.

Eine Variable behält so lange ihren Wert, bis er geändert oder gelöscht wird.

Für Integervariablen wird an den Variablennamen ein „%"-Zeichen angehangen.

Für Zeichenreihen-(String-)Variablen wird an den Variablennamen ein „$"-Zeichen angehängt.

Nachdem wir nun einige Möglichkeiten des Direktmodus ausgeschöpft haben, kommen wir zur Programmerstellung.

4/2.2

Eingabe — Verarbeitung — Ausgabe

Im vorliegenden Kapitel wollen wir die Grundzüge eines jeden Programmes darstellen. Jedes noch so große Programm beginnt mit dem Erfassen der zu verarbeitenden Daten, die in einem weiteren Schritt verarbeitet werden, um anschließend in neuer Form angezeigt zu werden. Bei größeren Programmen ist diese Vorgehensweise, wie sie in Bild 4/2.2-1 dargestellt ist, vielfach ineinander verschachtelt. Trotzdem weist jedes noch so kleine oder noch so große Computerprogramm diese Merkmale auf.

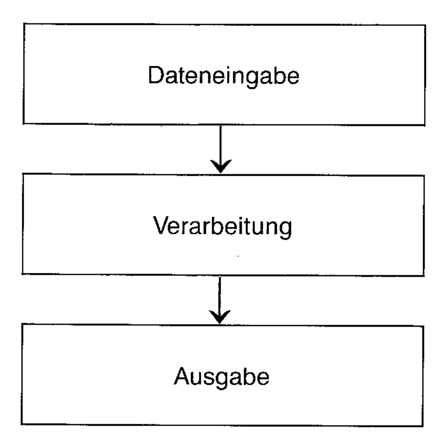


Bild 4/2.2-1: Grundstruktur aller Programme

4/2.2.1

Allgemeine Programmstruktur

In Basic erhält jede Programmanweisung eine Zeilennummer. Aufgrund dieser Zeilennummer wird das Programm von der kleinsten Zeilennummer bis zur größten Zeilennummer abgearbeitet, sofern es nicht auf Programmstrukturbefehle trifft, wie wir sie im nächsten Kapitel besprechen werden. Üblicherweise werden die Zeilennummern mit einer Schrittweite von zehn Zeilen gewählt, um auch spätere Einfügungen in das Programm möglich zu machen.

Merke: **Jede Programmzeile beginnt mit einer Zeilennummer. Das Programm wird in der Reihenfolge der Zeilennummer abgearbeitet.**

In der Regel endet das Programm mit dem sogenannten END-Befehl, dieser kann jedoch weggelassen werden, sofern sich das Programmende auch bei der letzten Zeilennummer befindet. Bei größeren Programmen ist dies selten der Fall.

Merke: **Mit dem END-Befehl wird ein Programm beendet.**

Eine weitere Möglichkeit des Programmabbruchs ist die Verwendung der RUN/STOP-Taste. Im Augenblick des Niederdrückens der Taste wird das Programm sofort abgebrochen, wobei das Programm noch bekanntgibt, in welcher Zeilennummer es abgebrochen wurde, wie wir später noch sehen werden.

4/2.2.2

Datenausgabe mit PRINT

Eigentlich ist es nichts neues, die Datenausgabe mit PRINT. Das im Direktmodus von uns verwendete „?“ ist nichts anderes als eine Abkürzung für den PRINT-Befehl. Schreiben Sie also noch mal

```
A=100 (RETURN-Taste)
```

und geben Sie das Quadrat von A aus mit

```
?A * A
```

Benutzen Sie nun anstelle des Fragezeichens den PRINT-Befehl und geben Sie ein

```
PRINT A * A (RETURN-Taste)
```

Das Ergebnis ist das gleiche. Nebenbei bemerkt, können Sie auch bei der Programmeditierung, zu der wir gleich kommen werden, statt eines PRINT ein „?“ eingeben. Der Computer macht dann selbständig aus dem „?“ ein PRINT.

Merke: Mit dem PRINT-Befehl können Daten ausgegeben werden.

Zuvor jedoch noch etwas zu den Variablenzuweisungen, die wir bis jetzt einfach so eingetippt haben. Wenn Sie eingeben „A=100“, so wird der Variablen A der Wert 100 **zugewiesen.** Diesem Umstand haben wir bisher keine weitere Bedeutung zugemessen, jedoch wollen wir nicht unerwähnt lassen, daß es sich in diesem Falle nicht um ein Gleichheitszeichen im mathematischen Sinne handelt, sondern um die sogenannte Zuweisung. Das Gleichheitszeichen sollte immer in der Bedeutung „ergibt sich zu“ gelesen werden. Unsere Eingabe bedeutet also nicht „A gleich 100“, sondern „A“ ergibt sich zu 100. Genaugenommen müßte man im Computer schreiben

```
LET A=100
```

Dieses LET wird aber von den meisten Programmierern der Einfachheit halber weggelassen. Die normale Schreibweise, z.B. bei der Erstellung von Flußdiagrammen, ist: A := 100.

Merke: **Mit dem „="-Zeichen können Variablenwerte zugewiesen werden.**

Daß den Variablen nicht nur konstante Zahlen zugewiesen werden können, sondern auch die Werte anderer Variablen, werden wir in unserem ersten kleinen Programmbeispiel gleich kennenlernen. Besonders dieser Umstand ist es, der den Computer zu einem leistungsfähigen Hilfsmittel macht.

Wie vorher erwähnt, bestehen Programmzeilen aus der Anweisung und der vorangestellten Zeilennummer. Wir tippen also ein:

```
10 A=125
20 B=A
30 PRINT A * B
```

Damit haben wir unser erstes kleines Basicprogramm fertiggestellt, bei dem in Zeile 10 der Variablen A der Wert 125 zugewiesen wird, in Zeile 20 der Variablen B der aktuelle Wert von A (ebenfalls 125) und in Zeile 30 das Produkt der beiden Zahlen ausgegeben wird.

Natürlich tut sich noch nichts auf dem Bildschirm, da wir uns im Editiermodus befinden und nur das Programm zeilenweise dem Rechner übergeben haben. Nach Abschluß einer jeden Zeile die RETURN-Taste zu drücken, diese Angaben wollen wir jedoch im folgenden weglassen. Vom Editiermodus in den Programm-/Rechenmodus kommen wir mit einem weiteren Basic-Befehl.

Merke: **Mit dem Befehl RUN wird das im Speicher befindliche Programm bei der ersten Zeilennummer gestartet.**

Den END-Befehl haben wir in unserem Beispiel weggelassen. Wir geben also nun RUN ein, und unter unserer Eingabe erscheint das Ergebnis: 15625. Gegenüber dem Direktmodus haben wir bis jetzt noch nicht viel gewonnen. Betrachten wir ein neues Beispiel.

Bevor wir jedoch ein neues Programm eingeben können, müssen wir das bisherige Programm löschen. Es befindet sich im Speicher und würde nur die Entwicklung unseres neuen Programmes stören. Daß es sich im Speicher befindet, können Sie ganz einfach feststellen, indem Sie eingeben

```
LIST
```

und natürlich das Drücken der RETURN-Taste nicht vergessen. Unsere zuvor eingetippten Zeilen erscheinen nochmals auf dem Bildschirm.

Merke: **Mit dem LIST-Befehl kann man das im Speicher befindliche Programm auf dem Bildschirm darstellen.**

Aber wir wollten ja unser Programm löschen. Auch hierzu gibt es einen eigenen Befehl: NEW

Merke: **Mit dem NEW-Befehl wird das im Speicher befindliche Programm gelöscht.**

Dieser Befehl ist nicht zu verwechseln mit dem CLR-Befehl, der nur die Variablen löscht. Nachprüfen können Sie den Vorgang, indem Sie nach einem NEW nochmals LIST eingeben.

Das soeben eingetippte Programm erscheint nicht mehr auf dem Bildschirm.

Nehmen wir an, wir wollen alle Zahlen von 1 bis 50 mit dem Faktor 2,25 multiplizieren. Auf den Direktmodus soll verzichtet werden, d.h. die Berechnungen sollen im Programm-Modus durchgeführt werden. Zwei Möglichkeiten bieten sich an:

```
10 A=1
20 PRINT A * 2.25
```

oder

```
10 A=1
20 B=2.25
30 PRINT A * B
```

Ihr Commodore 128 hat einen sogenannten bildschirmorientierten Editor. Nachdem Sie das Programm zum erstenmal mit RUN (RETURN) gestartet haben, erscheint das Ergebnis für 1 * 2.25. Wollen Sie nun das Ergebnis für die Zahl 2 berechnen lassen, so gehen Sie mit dem Cursor in die Programmzeile 10, so daß sich der Cursor auf der 1 befindet. Ist das Programm durch Manipulation mit anderen Tasten nicht mehr auf dem Bildschirm, so hilft hier wieder der LIST-Befehl. Nachdem der Cursor sich nun genau über der „1“ befindet, drücken Sie die Ziffer „2“. Die Änderung ist im Moment noch nicht an das Programm im Rechner übergeben worden, sondern steht nur auf dem Bildschirm. Dies passiert erst nach Drücken der RETURN-Taste oder auch der ENTER-Taste beim C 128.

Merke: **Programmzeilen lassen sich sehr leicht am Bildschirm ändern. Dazu ist die zu ändernde Stelle mit dem Cursor anzufahren und zu ändern. Anschließend muß die RETURN-Taste gedrückt werden. Auch die Taste INST/DEL kann als Hilfe zum Löschen oder Einfügen von Zeichen benutzt werden.**

Auch das Eintippen des RUN können wir uns ersparen, indem wir den Cursor auf den Befehl (irgendwo auf der Zeile) bewegen und die RETURN-Taste drücken. Das Ergebnis der neuen Berechnung ist 4.5. Sofern Sie an Ihrem Bildschirm eine 4.55 sehen, so kommt dies durch die Kombination des ersten Ergebnisses mit dem neuen Ergebnis. Sie sollten das alte Ergebnis zunächst durch Überschreiben mit Leerzeichen löschen. Auch können Sie das Programm an einer anderen Stelle auf dem Bildschirm starten (Eingabe von Run in einer anderen Zeile).

Trotz dieser einfachen Änderungsmöglichkeiten ist es natürlich sehr mühsam, das Ergebnis für eine Zahl von 1 bis 50 durchzurechnen. Die Änderung der Programmzeile können wir uns mit einem weiteren Befehl ersparen, den wir im nächsten Unterkapitel vorstellen wollen.

Zunächst soll jedoch noch auf ein paar weitere Besonderheiten des PRINT-Befehls eingegangen werden. Als ordnungsliebender Anwender wollen Sie natürlich nicht nur das Ergebnis der Rechnung betrachten, sondern auch die daran mitwirkenden Faktoren. Haben Sie unser zweites Beispiel (A=2, B=2.25) gewählt, so können Sie Zeile 30 sehr einfach durch Überschreiben ändern in

```
30 PRINT A,B,A * B
```

Hier haben Sie einen weiteren Umstand kennengelernt, der Ihnen bei der Editierung Ihrer Programme sehr hilfreich sein kann.

Merke: **Programmzeilen können vollständig neu eingegeben werden, dabei wird die alte Programmzeile überschrieben (gelöscht).**

Wenn Sie nun das geänderte Programm starten, werden Sie feststellen, daß die drei Daten nebeneinander auf dem Bildschirm erscheinen. Das Komma zwischen den Variablen wirkt dabei als Tabulator. Die erste Zahl steht aber nicht am linken Bildschirmrand, sondern ein Zeichen weiter hinten. Dies kommt durch das — in unserem Falle positive — Vorzeichen. Bei negativen Zahlen steht hier ein „-“.

Wenn Sie nun — unter Zuhilfenahme des Cursors — die Zwischenräume zwischen den ersten Ziffern einer jeden Zahl ausrechnen, so werden Sie feststellen, daß die ersten Ziffern jeweils zehn Zeichenpositionen auf dem Bildschirm voneinander entfernt sind. Der Tabulatorabstand beträgt also zehn Bildschirmzeichen.

Merke: **Beim PRINT-Befehl können diverse Ausgaben (alle Variablentypen und Konstantenwerte) durch „ , “ voneinander getrennt werden. Die Ausgabe erfolgt jeweils im Abstand von zehn Bildschirmzeichen. Bei längeren Ausgaben ist dies ein Vielfaches von zehn Bildschirmzeichen.**

Ersetzen wir nun die Kommata in Zeile 30 durch ein „ ; “ wie folgt

```
30 PRINT A;B;A * B
```

Dies können Sie sowohl durch Überschreiben der Kommata mit einem „ ; “ — anschließend das RETURN drücken nicht vergessen — oder durch Neueingabe der Zeile 30. Wenn Sie das Programm starten, werden Sie feststellen, daß die Ziffern nur durch ein Leerzeichen — und die Stelle für das Vorzeichen — voneinander getrennt sind.

Merke: **Im PRINT-Befehl können Variablen und Konstanten durch „ ; “ getrennt ausgegeben werden. Die Ausgabe erfolgt ohne Tabulatorposition wie bei „ , “. „ , “ und „ ; “ können innerhalb eines PRINT-Befehles gemischt werden.**

Diese beiden Hilfsmittel sind sehr wichtig für die spätere anwenderfreundliche Ausgabe bei Ihren Programmen. An dieser Stelle ist es vielleicht angeraten, die Lektüre des Buches zwecks Übung nach eigenen Vorstellungen zu unterbrechen.

4/2.2.3

Datenerfassung mit INPUT

Wem das Ändern von Programmzeilen bei fortlaufenden Berechnungen zu kompliziert erscheint, der kann den aktuellen Wert auch per Programm vom Bildschirm erfassen lassen. Dazu dient uns der INPUT-Befehl. Wir ersetzen Zeile 10 durch

```
10 INPUT A
```

Wenn Sie nun das Programm starten, erscheint als erstes ein Fragezeichen, und dahinter blinkt der Cursor. Der Rechner erwartet nun eine Eingabe durch Sie, nämlich den gewünschten Wert für die Variable A. Geben Sie jetzt eine 3 ein, und drücken die RETURN-Taste, so erscheint — bei der letzten Form des Programmes aus dem vorigen Kapitel:

```
3  2.25  6.75
```

Wenn Sie erneut das Programm mit RUN starten, können Sie den nächsten Wert eingeben usw.

Merke: **Mit dem INPUT-Befehl können Werte für Variablen vom Bildschirm erfaßt werden.**

Für unseren einfachen Fall mag das „?" als Hinweis für eine Dateneingabe ausreichen. Bei komplizierten Anwenderprogrammen ist es manchmal unmöglich, aber auf jeden Fall anwenderunfreundlich, festzustellen, welche Eingabe der Computer im Augenblick erwartet. In diesem Falle wird der INPUT-Befehl weiter modifiziert, indem wir eine Meldung für den Anwender einbauen. Dies könnte wie folgt aussehen:

```
10 INPUT"AKTUELLER WERT";A
```

Wenn wir nun das Programm starten, erscheint als Meldung des Computers

```
AKTUELLER WERT?
```

Dahinter blinkt wieder der Cursor und erwartet unsere Eingabe.

Merke: **Innerhalb der Input-Anweisung kann ein Anwenderhinweis (Text) durch „ " " begrenzt werden. Zwischen dem letzten Anführungszeichen des Textes und der Variable ist unbedingt ein „ ; " einzugeben.**

Auch unsere Variable B können wir mittels des INPUT-Befehls erfassen. Dies als Übung für Sie. Ändern Sie Zeile 20 entsprechend ab.

Aber auch innerhalb eines INPUT-Befehls können mehrere Variablenwerte erfaßt werden. Die Variablen werden dabei durch „ , " getrennt.

Merke: **Nach INPUT können mehrere Variablen durch Kommata getrennt, eine sogenannte Variablenliste, erfaßt werden.**

Unsere Programmzeile könnte dann wie folgt aussehen

```
10 INPUT AKTUELLE WERTE FÜR A/B";A,B
```

Jetzt stört natürlich noch Zeile 20, da diese überflüssig ist. Dazu geben wir lediglich die Ziffer 20 am Zeilenanfang einer leeren Zeile ein und drücken die RETURN-Taste. Wir merken uns also:

Merke: **Durch Eingabe einer Zeilennummer (ausschließlich) und Drücken der RETURN-Taste wird eine Programmzeile gelöscht.**

Dies können Sie sich mit dem LIST-Befehl sofort veranschaulichen. Daß nun die Zeile 20 in unserem Programm fehlt, ist für die Bearbeitung des Programmes nicht weiter tragisch. Wenn Sie nun das Programm starten, erscheint die Meldung

```
AKTUELLE WERTE FUER A/B?
```

Geben Sie die beiden Werte, z.B. 5 und 95, durch Komma getrennt ein, und drücken Sie dann die RETURN-Taste. Das Ergebnis erscheint in der üblichen Form. Geben Sie aber z.B. nur die 5 als Wert für A ein und drücken Sie dann die RETURN-Taste, so erscheinen in der nächsten Zeile zwei aufeinanderfolgende Fragezeichen. Damit deutet der Computer Ihnen an, daß er noch weitere Eingaben erwartet, die sich innerhalb des gleichen INPUT-Befehls befinden. Wenn Sie nun die 95 eingeben, erhalten Sie das gleiche Ergebnis wie vorhin.

Merke: **Bei einem INPUT-Befehl mit Variablenliste müssen alle Werte für die Variablen durch Kommata getrennt eingegeben werden. Andernfalls erfragt der Rechner durch „??“ so lange Werte, bis jeder Variablen ein Wert zugewiesen ist. Sinnvollerweise sollte ein Hinweis im Text des INPUT-Befehls nicht fehlen.**

Wir haben bewußt für Sie einfache Beispiele herangezogen, da z.B. komplizierte mathematische Berechnungen im Thema nicht weiterführen. Trotzdem sollten Sie an dieser Stelle Ihre Phantasie etwas spielen lassen oder vielleicht auch ein aktuelles Problem aufgreifen, um das Dargestellte zu üben.

Im nächsten Kapitel wollen wir dann einige Programmstruktur-Befehle durchsprechen, die Ihnen u.a. den lästigen Neustart des Programmes abnehmen, wenn Sie z.B. die Vorgabe, alle Zahlen von 1 bis 50 mit 2.25 zu multiplizieren, erfüllen wollen.

4/2.3

Programmstruktur-Befehle

In Kapitel 4/2.2 haben wir die grobe Programmstruktur „Eingabe — Verarbeitung — Ausgabe“ kennengelernt und dies an einem Minimalprogramm verdeutlicht. Hier noch ein weiteres Beispiel für diese Vorgehensweise.

```
10 INPUT"FAKTOREN FUER A*B"; A,B
20 C=A*B
30 PRINT C
```

In diesem Programm stellt jede der Programmzeilen einen der oben genannten Schritte dar:

```
10 Datenerfassung
20 Verarbeitung
30 Datenausgabe
```

Wollte man nun z.B. die Produkte für alle Zahlen von 1 bis 100 jeweils sowohl für den Wert von A als auch für den Wert von B errechnen, müßte man zehntausendmal das Programm mit RUN und der RETURN-Taste neu starten. Dies wollen wir uns als erstes in diesem Kapitel vereinfachen.

4/2.3.1

Springen mit GOTO

Wir ergänzen vorstehendes Programm um folgende Zeile:

```
40 GOTO 10
```

Vergessen Sie nicht den Druck auf die RETURN-Taste. Vielleicht sollten Sie sich nun auch das komplette Programm mit dem LIST-Befehl anschauen.

Was besagt nun unsere neue Zeile 40. Das GOTO — man kann es übersetzen mit „gehe nach" — beinhaltet einen sogenannten unbedingten Programmsprung. Kommt das Programm zu unserer neuen Zeile 40, so erhält es die Anweisung in Zeile 10 fortzufahren. Wenn man sich unser Programm nun genau betrachtet, ist es eine prinzipielle Endlosschleife, da es bei Zeile 40 immer wieder in Zeile 10 beginnt und von dort aus wieder zur Zeile 40 gelangt. Dies sind die berühmt-berüchtigten Endlosschleifen, die wir hier bewußt in Kauf nehmen.

Merke: **Mit GOTO und Angabe einer Zeilennummer wird das Programm veranlaßt, an der angegebenen Zeilennummer fortzufahren. Ist die angegebene Zeilennummer im Programm nicht vorhanden, so meldet der Rechner einen »?UNDEF'D STATEMENT ERROR« mit Angabe der Zeilennummer, in der der Sprungbefehl steht.**

Der GOTO-Befehl hat das Lager der Programmierer gespalten. Die einen befürworten die Anwendung dieses Befehles, die anderen lehnen ihn als unsaubere Programmierung ab. Obwohl im Sinne korrekter und übersichtlicher Programmierung der letzteren Gruppe zuzustimmen ist, ist dem Anfänger mit dem GOTO-Befehl ein einfaches Hilfsmittel in die Hand gegeben, den Programmverlauf nach seinen Wünschen zu beeinflussen. Die anderen möglichen Verfahren sind an dieser Stelle viel zu aufwendig. Sofern Sie sich schon eine Meinung zu diesem Thema gebildet haben und auf GOTO verzichten wollen, können Sie in unserem speziellen Falle Zeile 40 ersetzen durch:

```
40 RUN
```

Damit wird das Programm veranlaßt, als letzten Befehl sich selbst wieder aufzurufen — auch dies ist möglich. Dies gilt jedoch nur in unserem speziellen Fall und dürfte die Ausnahme zur Umgehung des GOTO-Befehls sein.

Wenn Sie nun das Programm aufrufen, wird jedesmal nach der Ausgabe des Ergebnisses erneut nach einer Eingabe gefragt. Wie gesagt, befinden wir uns in einer Endlosschleife. Da man das Programm mit der RUN/STOP-Taste beim Warten des Computers auf eine Eingabe nicht abbrechen kann, müssen wir uns mit einem kleinen Trick behelfen. Drücken Sie zunächst die RETURN-Taste und halten Sie diese fest. Drücken Sie anschließend die RUN/STOP-Taste. Für Neugierige: Auch die RETURN-Taste hat eine Dauerfunktion. Drücken Sie z.B. nur diese Taste, so erscheint nach der Abfrage jeweils als Ergebnis eine „0". Der Rechner hat als Eingabe einfach die „0" angenommen, wodurch das Ergebnis auch null ist. Durch das Drücken der RETURN-Taste haben wir aber bewirkt, daß das Programm die Zeile 10 verläßt. In den anderen Zeilen ist die RUN/STOP-Taste aber wirksam.

Merke: Mit der RUN/STOP-Taste kann ein Programmlauf abgebrochen werden. Dies funktioniert nicht, während der Rechner auf eine Eingabe wartet.

4/2.3.2

Bedingungen stellen mit IF . . . THEN

Entfernen wir nun den INPUT-Befehl in Zeile 10 aus unserem Programm und schreiben wir statt dessen

```
10 A=1
11 B=1
35 A=A+1
```

Wir haben also den INPUT-Befehl wieder durch eine direkte Zuweisung an die Variablen A und B in den Zeilen 10 und 11 ausgetauscht und in Zeile 35 weisen wir der Variablen A den eigenen Wert vermehrt um 1 zu.

Merke: Bei einer Zuweisung kann auf beiden Seiten des Gleichheitszeichens die gleiche Variable vorkommen. Während der Zuweisung, d.h. auf der rechten Seite der Zuweisung, hat die Variable noch ihren alten Wert. Nach Abschluß der Anweisung hat die Variable den neu zugewiesenen Wert.

Wenn wir nun unser Programm starten, wird die linke Seite des Bildschirms fortlaufend mit 1-en gefüllt. Wir ändern die Zeilen 30 und 40 noch so ab, daß wir wieder alle Variablenwerte in Zeile 30 ausgegeben bekommen, in Zeile 40 aber zur Zeile 20 springen, d.h. die Zuweisungen in den Zeilen 10 und 11 nach dem GOTO nicht mehr berücksichtigt werden.

```
30 PRINT A,B,C
40 GOTO 20
```

Wenn wir nun das Programm starten, werden drei Spalten auf dem Bildschirm sichtbar, wobei die äußeren Spalten jeweils die natürlichen Zahlen fortlaufend darstellen und die mittlere Spalte konstant 1-en enthält. Brechen Sie nun das Programm mit der RUN/STOP-Taste ab und machen Sie es mit dem LIST-Befehl wieder auf dem Bildschirm sichtbar.

Unser Ausgangsproblem in diesem Unterkapitel war es ja, sowohl für A als auch für B die Werte 1 bis 100 annehmen zu lassen, und diese miteinander zu multiplizieren. Nach längerem Lauf des Programms ist der Wert der Variablen A weit über die 100 hinausgeschossen, während der Wert der Variablen B immer noch bei 1 liegt. Dies wollen wir jetzt ändern.

Umgangssprachlich ausgedrückt würde man sagen, das Programm soll folgendes für uns tun: *Wenn* A den Wert 100 angenommen hat, *dann* soll der Wert in B um 1 erhöht und der Wert in A auf 1 zurückgesetzt werden. Nebenbei kommen wir jetzt dem Sinn eines Computers wieder etwas näher, da seine Hauptaufgabe als Rechner darin liegt, viele gleichartige Operationen mit unterschiedlichen Werten auszuführen, die in der Regel viel komplizierter als unser Beispiel sind.

Unser „wenn . . ., dann . . .“ würde man in englisch ausdrücken mit „IF . . . THEN“ und genauso wird es im Computer auch gemacht.

Merke: **Mit der Befehlskombination IF . . . THEN . . . lassen sich bedingte Anweisungen realisieren. Nach IF steht der sogenannte Bedingungsteil, und nach THEN die Anweisungen, die ausgeführt werden sollen, wenn die Bedingung erfüllt ist.**

Wir fügen in unser Programm also noch zwei weitere Zeilen ein:

```
36 IF A=101 THEN B=B+1
37 IF A=101 THEN A=1
```

Da diese komplizierte Formulierung im Rechner zuviel Speicherplatz braucht und auch noch andere Fallstricke aufweist (probieren Sie es durch Vertauschen der Zeilen 37 und 36, dann wird B nie erhöht) gibt es ein weiteres programmtechnisches Hilfsmittel.

Merke: **Innerhalb einer Zeile können mehrere Anweisungen programmiert werden. Die Anweisungen sind durch ein »:« zu trennen. Im THEN-Teil einer Bedingung werden alle Anweisungen nur durchgeführt, wenn die Bedingung erfüllt ist.**

Wir können also auch schreiben:

```
36 IF A=101 THEN B=B+1 : A=1
```

Die Zeile 37 kann wie üblich (Zeilennummer und RETURN-Taste) gelöscht werden. Wenn Sie nun das Programm starten, werden Sie feststellen, daß in der ersten Spalte (Werte der Variablen A) nur die Zahlen 1 bis 100 durchlaufen werden und bei jedem Umsetzen des Wertes A von 100 auf 1 der Wert in B um 1 erhöht wird. Damit die Wartezeit vor dem Bildschirm nicht zu lang wird, werden wir nun die Zeile 36 etwas ändern. Gehen Sie mit dem Cursor auf die zweite 1 von „101“ d.h. auf die Einerstelle. Drücken Sie dann die Taste DEL und anschließend die RETURN-Taste. Wenn Sie das Programm nun neu starten, nimmt die Variable A nur Werte von 1 bis 10 an. Der Wert von B erhöht sich fortlaufend, bis in alle Ewigkeit, wenn Sie das Programm nicht mit der RUN/STOP-Taste abbrechen. Hier lernen wir einen weiteren Befehl kennen.

Merke: Trifft das Programm auf einen STOP-Befehl, so bricht es ab, wobei die Zeilennummer beim Abbruch angegeben wird.

Wir ergänzen also unser Programm um die Zeile 37 wie folgt:

```
37 IF B=11 THEN STOP
```

(RETURN-Taste nicht vergessen). Beachten Sie bitte auch, daß bei den Bedingungen in Zeile 36 und 37 das Gleichheitszeichen nicht als Zuweisung, sondern im Sinne einer mathematischen Gleichheit behandelt wird. Wenn Sie nun das Programm starten, enthält die letzte Ausgabezeile die Werte 10,10,100 und es erscheint die Meldung

```
BREAK IN 37
READY.
```

Dies ist natürlich keine sehr saubere Programmierung, was wir aber gleich ändern wollen. Statt der Anweisung STOP kann auch die Anweisung END gegeben werden, die wir schon kennen. Da wir aber noch eine weitere Möglichkeit der bedingten Anweisung aufzeigen wollen, fügen wir die Zeile 50 wie folgt ein

```
50 END
```

und ändern unsere Zeile 37

```
37 IF B=11 THEN GOTO 50
```

Merke: Als Anweisung im THEN-Teil einer bedingten Anweisung kann auch ein GOTO »Zeilennummer« stehen. Das GOTO kann der Einfachheit halber auch weggelassen werden. Eine weitere Möglichkeit ist das Ersetzen des THEN durch das GOTO.

Durch Änderung der Zeile 37 sollten Sie diese Angaben überprüfen.

An dieser Stelle wird auch deutlich, warum der GOTO-Befehl als unbedingter Sprungbefehl bezeichnet wird: Mit IF . . . THEN . . . sind bedingte Programmsprünge möglich.

Übrigens der mathematische Hintergrund zu unserem kleinen Beispielprogramm: Werden die Werte für die Variable A und B als Seitenlängen eines Rechtecks angesehen, so ist in der Variablen C der Flächeninhalt wiedergegeben. Im folgenden Kapitel wollen wir unser Programm noch etwas vereinfachen und die Fläche eines Rechtecks bei konstanter Länge einer Seite berechnen.

4/2.3.3

Programmschleifen mit FOR . . . NEXT

Im Folgenden wollen wir mehrere Programmversionen für ein Beispiel aufzeigen. Berechnet werden soll die Fläche eines Rechtecks, wobei die Länge der einen Seite konstant 10 beträgt und die Länge der anderen Seite zwischen 20 und 50 in ganzzahligen Schritten aufwärts betragen soll. Löschen Sie dazu mit NEW das zuletzt im Rechner befindliche Programm und geben Sie ein:

```
10 A=10
20 B=20
30 PRINT A,B
40 C=A*B
50 PRINT C
60 IF B 50 THEN B=B+1 : GOTO 30
```

Zunächst lernen wir in Zeile 30 eine weitere Möglichkeit des PRINT-Befehles kennen.

Merke: **Die zur Trennung zwischen Variablenwerten beim PRINT-Befehl verwendeten »,« und »;« können auch am Ende eines PRINT-Befehles angegeben werden. Dadurch wird der Zeilenvorschub nach dem PRINT-Befehl unterdrückt und bei der nächsten Ausgabe wird an der Stelle fortgefahren, die durch das letzte Zeichen des vorherigen PRINT-Befehles gegeben ist.**

Wenn Sie nun das Programm starten, erhalten Sie wieder die Ausgabe in drei Spalten, wobei die erste Spalte nur den Wert 10 enthält, die zweite Spalte fortlaufende Werte von 20 bis 50 und die dritte Spalte das Ergebnis der Multiplikation.

Die Zeilen 30 bis 60 werden auch als Programmschleife bezeichnet, da sich diese Zeilen des öfteren wiederholen. Dabei ist bei jedem Durchlauf durch die Zeilen 30 bis 60 der Wert von B um eine Einheit größer als beim Durchlauf zuvor.

Bevor wir nun zu einer weiteren Vereinfachung dieser Programmform kommen, wollen wir anhand des dargestellten Programmes noch einige Begriffe klären:

Laufvariable:	Die Variable B kann in unserem Beispiel als Laufvariable bezeichnet werden, da sie die Werte von 20 bis 50 durchläuft.
Anfangswert:	Der Anfangswert der Programmschleife ist 20, da der Wert von B vor Eintritt in die Programmschleife den Wert 20 aufweist.
Schleifenendekriterium:	Durch Zeile 60 ist innerhalb der bedingten Anweisung das Endekriterium auf 50 festgelegt. Der Befehl „GOTO 30“ wird nur erreicht, solange der Wert von B kleiner als 50 ist.
Schrittweite:	Als Schrittweite wird der Wert bezeichnet, um den die Laufvariable — hier: B — bei jedem Schleifendurchlauf erhöht wird. In unserem Fall ist es eine Einheit.

Was liegt näher als diese Werte dem Computer in einer eigenen Zeile einzugeben und das Ende der Schleife durch einen gesonderten Befehl zu markieren. In einfachem Deutsch würde man sagen: „Mit der Variablen B tue von 20 bis 50 folgendes . . . Ende der Anweisungen“. So ähnlich akzeptiert es auch Ihr Rechner, nur versteht er leider kein Deutsch.

Merke: **Programmschleifen werden im Rechner mit »FOR ‚Laufvariable‘=‚Anfangswert‘ TO ‚Schleifenendekriterium‘ STEP ‘Schrittweite‘« definiert.**

Das Ende der Anweisungen einer Programmschleife wird durch den Befehl NEXT markiert.

Die Schrittweite wird bei jedem Schleifendurchlauf zum aktuellen Wert der Laufvariablen hinzugezählt. Soll sie abgezogen werden, so ist eine negative Schrittweite anzugeben. Die Schrittweite muß nicht

in ganzen Zahlen, sondern kann auch in Dezimalzahlen angegeben werden.

Programmschleifen können auch ineinandergeschachtelt werden, sofern verschiedene Variablennamen für die Laufvariablen vorgegeben werden.

Die Schrittweite kann auch weggelassen werden, sie beträgt dann 1.

Programmschleifen mittels FOR . . . NEXT sind die erste etwas kompliziertere Programmstruktur, die wir kennenlernen. Zur besseren Erläuterung wollen wir unser letztes Beispielprogramm wie folgt abändern:

```
20 FOR B=20 TO 50
60 NEXT
```

Die restlichen Zeilen bleiben unverändert. In unserem Beispiel haben wir die Schrittweite ausgelassen, da sie sowieso eine Einheit beträgt. Wenn Sie nun das geänderte Programm starten, werden Sie bei der Ausgabe keine Auswirkungen feststellen können. Was ist nun passiert?

Wenn der Rechner zum erstenmal auf die Zeile 20 trifft, so erhält die Variable B den Wert 20 zugewiesen. Außerdem merkt sich der Rechner — wie ist hier egal —, daß eine Programmschleife vorliegt. Die folgenden Anweisungen werden normal abgearbeitet, bis das Programm auf den Befehl NEXT trifft.

An dieser Stelle wird nun geprüft, ob die Laufvariable, die der Computer sich gemerkt hat, bereits den Wert des Endekriteriums — in unserem Fall 50 — überschritten hat. Wenn nicht, wird die Schrittweite zur Laufvariablen addiert und zur nächsten Anweisung nach dem FOR . . . -Befehl übergegangen. Trifft der Computer auf das NEXT und der Wert des Schleifenendekriteriums ist durch die Laufvariable überschritten, so fährt er mit der nächsten Anweisung nach dem NEXT fort. Durch die vorliegende Konstruktion wird die Programmschleife aber auf jeden Fall durchlaufen. Eventuelle Kollisionen, z.B. Anfangswert 10 und Endekriterium 5 werden erst bei Auftreffen auf NEXT festgestellt. Dies ist je nach Anwendungsfall vom Anwender durch geeignete Programmierung abzufangen.

Merke: Eine Programmschleife wird mindestens einmal durchlaufen.

Ändern Sie nun Zeile 20 so ab, daß die Schrittweite eine halbe Einheit beträgt:

```
20 FOR B=20 TO 50 STEP 0.5
```

Probieren Sie es auch mit anderen Schrittweiten. Als nächstes wollen wir auch die Variable A verschiedene Werte durchlaufen lassen, z.B. von 10 bis 30 mit Schritt-

weite 2. Dadurch können Sie überprüfen, daß Programmschleifen auch ineinandergeschachtelt werden können. Folgende Änderungen sind an unserem Beispielprogramm notwendig:

```
10 FOR A=10 TO 30 STEP 2
70 NEXT
```

Achtung: Das NEXT in Zeile 60 bezieht sich immer noch auf die Schleife zur Laufvariablen B, das NEXT in Zeile 70 bezieht sich auf die Laufvariable A. Wenn Sie diesen Tatbestand kenntlich machen wollen, können Sie auch eingeben

```
60 NEXT B
70 NEXT A
```

Generell verwendet der Computer ein NEXT immer für die aktuelle Schleife, in der er sich befindet. Am besten veranschaulichen Sie dies, indem Sie das Programm mit RUN starten. Zunächst erhält bei Programmbeginn die Variable A den Wert 10 zugewiesen und in der nächsten Zeile die Variable B den Wert 20. Dann wird die Schleife bis zur Zeile 60 durchlaufen und auf den Wert B die Schrittweite von 0.5 addiert und zur Zeile 20 übergegangen. Erst wenn die innere Schleife der Laufvariablen B beendet ist, tritt das NEXT in Zeile 70 in Kraft und der Wert der Laufvariablen A wird um zwei Einheiten erhöht. Dann wird wieder die sogenannte innere Schleife ab Zeile 20 bis zur Zeile 60 bearbeitet.

Auch an dieser Stelle sei wieder angeraten, das Beispielprogramm nach eigenen Wünschen zu modifizieren. Haben Sie z.B. ein NEXT zuviel eingegeben, so meldet der Rechner einen „NEXT WITHOUT FOR ERROR“.

4/2.3.4

Hilfsdienste aufrufen mit GOSUB . . . RETURN

In diesem Unterkapitel wollen wir unser letztes Beispielprogramm noch etwas abändern. Die Verwendung innerhalb des Beispieles ist zwar etwas untypisch, für den neuen Basicbefehl (bzw. die Befehlsgruppe), die Verwendung wird daran aber gut deutlich. Gemeint ist die Unterprogrammtechnik.

Was sind Unterprogramme? Unterprogramme sind generell Programmsequenzen, d.h. Folgen von Anweisungen, die an mehreren Stellen im Programm benötigt, aber nur einmal programmiert werden. Unterprogramme haben eine definierte Einsprungstelle, dargestellt durch eine Zeilennummer (Zeilennummer der ersten Anweisung im Unterprogramm). Das Ende eines Unterprogrammes wird ähnlich gekennzeichnet wie bei den Programmschleifen mit dem NEXT, jedoch wird hier zur Unterscheidung ein RETURN verwendet.

Aufgerufen wird ein Unterprogramm mit dem Befehl GOSUB „Zeilennummer". Ähnlich dem unbedingten Sprung mit GOTO wird sofort zu dieser Zeilennummer übergegangen und ähnlich zu den FOR . . . NEXT-Schleifen merkt sich der Rechner beim Aufruf eines Unterprogrammes, wo er das Unterprogramm aufgerufen hat. Beendet wird das Programm mit dem *Basic-Befehl* RETURN, was nicht zu verwechseln mit der RETURN-Taste ist.

Trifft der Rechner auf den Befehl RETURN, so springt er zu dem Befehl hinter dem aufrufenden GOSUB zurück. Im Gegensatz zum GOTO liegt also kein „Sprung ohne Wiederkehr" vor, sondern der Rechner kehrt nach Beendigung des Unterprogrammes an die sogenannte Aufrufstelle zurück.

Bevor wir jedoch unser Beispiel abändern, wollen wir noch einige weitere Begriffe klären. Obwohl das Basic gegenüber sogenannten blockorientierten Sprachen keine unterprogrammeigenen Variablen kennt, so können doch Variablen an das Unterprogramm übergeben oder daraus zurückgeführt werden. Diese Variablen werden dann je nach Verwendungszweck auch als Eingabeparameter oder Ausgabeparameter bezeichnet.

Ein Unterprogramm aus unserem letzten Beispiel haben wir nun schnell kreiert. Löschen Sie zunächst Zeile 70 durch Eingabe der Zeilennummer und anschließendem Druck auf die RETURN-Taste. Geben Sie anschließend dieses NEXT mit der Zeilennummer 18 ein und definieren Sie ein Programmende in Zeile 19 wie folgt:

```
10 NEXT
19 END
```

Wenn Sie sich nun das gesamte Programm mit LIST anschauen, wird eine leere Schleife in den Zeilen 10 und 18 durchlaufen und in Zeile 19 das Programm beendet. Starten Sie das Programm, so sehen Sie nichts auf dem Bildschirm. Geben Sie nun in Zeile 70 den RETURN-Befehl ein

```
70 RETURN
```

und fügen Sie Zeile 15 wie folgt ein

```
15 GOSUB 20
```

Wenn Sie das Programm starten, erhalten Sie als Ausgabe exakt das gleiche wie bei dem ungeänderten Beispielprogramm.

Was passiert? Das Programm läuft in der festgelegten Schleife von Zeile 10 bis Zeile 18. In Zeile 15 erfolgt der Unterprogrammaufruf, der die einzige Anweisung innerhalb der Schleife bildet. Beim Auftreffen auf diesen Unterprogrammaufruf setzt der Rechner das Programm in Zeile 20 fort und durchläuft unsere Beispielschleife mit der Laufvariablen B. Ist diese Schleife beendet, so trifft der Rechner auf den Befehl RETURN und geht zu dem Befehl über, der auf „GOSUB 20" folgt. Dies ist das Endekriterium für die Programmschleife mit der Laufvariablen A und der Rechner geht wieder zur Zeile 10 mit erhöhter Laufvariable A zurück, um dann erneut das Unterprogramm ab Zeile 20 in Zeile 15 aufzurufen.

Merke: Unterprogramme werden in Basic mit GOSUB »Zeilennummer« und abschließendem RETURN definiert.

Unterprogramme können genau wie Programmschleifen ineinandergeschachtelt werden.

Programmschleifen und Unterprogramme können ebenfalls geschachtelt werden, wobei Programmschleifen, die außerhalb eines Unterprogramms beginnen und innerhalb eines Unterprogramms enden, unzulässig sind.

Versuchen Sie nun, ein Beispielprogramm zu rechnen, indem für einen Quader mit den Seiten A, B und C die Oberfläche, die Kantenlänge und das Volumen berechnet wird. Schreiben Sie für die Berechnung der Oberfläche, des Volumens und der Kantenlänge jeweils ein eigenes Unterprogramm und lassen Sie die Kante A mit Werten von 10 bis 50 mit Schrittweite 5, die Kante von B mit Werten von 60 bis 100 mit Schrittweite 10 und die Kante C von 30 bis 40 mit Schrittweite 1 laufen. Sicher wird dies nicht gleich auf Anhieb gelingen. Trotzdem sollten Sie nicht gleich zur Lösung übergehen, sondern das Buch erstmal zur Seite legen.

Hier die Lösung:

```
100 REM ------------------------------
110 REM ---    BEISPIELPROGRAMM      ---
120 REM ------------------------------
130 :
140 FOR A=10 TO 50 STEP 5
150 : FOR B=60 TO 100 STEP 10
160 : : FOR C=30 TO 40
170 : : : GOSUB 1000
180 : : : GOSUB 2000
190 : : : GOSUB 3000
200 : : : PRINTA,B,C,O,K,V
210 : : NEXT
220 : NEXT
230 NEXT
240 :
1000 REM ---        OBERFLAECHE      ---
1010 :
1020 O=2*A*B + 2*A*C + 2*B*C
1030 RETURN
1040 :
2000 REM ---        KANTEN           ---
2010 :
2020 K=2*A + 2*B + 2*C
2030 RETURN
2040 :
3000 REM ---       VOLUMEN           ---
3010 :
3020 V=A*B*C
3030 RETURN
```

Listing 4/2.3.4

4/3
Basic 7.0

Die eingebaute Programmiersprache des C 128 ist, ebenso wie beim C 64, Basic. Ein entscheidender Vorteil des C 128 ist das Mehr an Basic-Befehlen, insbesondere im Bereich der Grafik-Programmierung, der Sound-Programmierung und der Disketten-Befehle. Zur Unterscheidung vom Basic des C 64 wird beim Basic des C 128 als Versionsnummer 7.0 als Zusatz benutzt.

Anders als in Kapitel 4/2 wollen wir hier nicht direkt einen Kurs für Basic 7.0 anbieten, sondern zunächst einen kurzen Kommentar zu jedem Befehl mit einigen kleinen Anwendungsbeispielen geben. Wer die Grundkenntnisse in Basic beherrscht (siehe Kapitel 4/2), kann die neuen Befehle sicherlich weitgehend aufgrund der Syntax (wie wird der Befehl verwendet) und Semantik (was macht der Befehl) einsetzen.

4/3.1
Kommentar zu den Befehlen mit kleinen Anwendungsbeispielen

Zur Anwendung von Programmbefehlen reicht meist die Kenntnis der Syntax und Semantik aus. Trotzdem wollen wir versuchen, Sie innerhalb dieses Kapitels einerseits vor einigen Klippen zu warnen und andererseits auf ergänzende Möglichkeiten der Befehle hinweisen.

Dazu beschäftigen wir uns zunächst mit den Programmierhilfen, dem Handwerkszeug eines jeden Programmierers. Es folgt eine kommentierte Übersicht über die Grafik- und Sound-Befehle, sowie Befehle zum Handling einer Floppy.

An dieser Stelle wollen wir auch auf Kapitel 4/4 hinweisen, wo die Themen Grafik und Sound in Kursform an einprägsamen Beispielen beschrieben werden. Hier wollen wir nur auf die Möglichkeiten und Grenzen der Befehle eingehen. Für die Floppy-Befehle sei auch auf Kapitel 4/4.7 (Dateiverwaltung) hingewiesen, wo wir eine Dateiverwaltung grundlegend erarbeiten wollen.

4/3.1.1

Programmierhilfen

Bevor man die Programmierarbeiten an irgendeinem Rechner beginnt, sollte man sich zunächst mit den zur Verfügung stehenden Hilfsmitteln der Programmiersprachen und des Betriebssystems vertraut machen. Beim C 128 ist für den Anwender im Prinzip beides nicht zu trennen, da sich die Basic-Befehle kaum von den System-Kommandos unterscheiden. Die Tabelle in Kapitel 11/1.1 gibt jedoch über diesen Unterschied Auskunft, wobei hier auch noch zwischen Befehlen und Funktionen unterschieden wird.

Die Befehle innerhalb dieses Kapitels werden wir in zwangloser Weise aneinanderreihen, so wie sie sich beim Einarbeiten in den Rechner darstellen würden. Beginnen wir mit einem der ersten Hilfsmittel, das insbesondere bei der Programmeditierung wichtig ist.

4/3.1.1.1

AUTO — fortlaufende Zeilennumerierung bei der Programmeditierung

Besonders beim Abtippen von Programmen aus Zeitschriften oder Büchern — was meist zu Beginn einer Bekanntschaft mit einem neuen Rechner zu empfehlen ist —

ist es recht mühselig, jeweils die nächste Zeilennummer in den Rechner einzutippen, zumal man sich immer vergewissern muß, welche Zahl die nächste ist.

Hier hilft der AUTO-Befehl, da er diese Arbeit für Sie erledigt. Nachdem Sie den AUTO-Befehl — z.B. mit der gebräuchlichsten Schrittweite 10 — aufgerufen haben, erscheint zunächst noch gar nichts auf dem Bildschirm, da erst dem Rechner gesagt werden muß, mit welcher Zeile er beginnen soll. D.h., daß Sie nicht immer beim Anfang eines Programmes anfangen müssen, sondern der AUTO-Befehl auch sehr hilfreich ist, wenn sie in ein bestehendes Programm einige Zeilen einfügen wollen. Für eine einzige Zeile ist es natürlich weniger sinnvoll.

Der AUTO-Befehl ist so lange wirksam, wie Sie eine Zeilennummer mit anschließendem Basic-Text eingeben. Geben Sie eine leere Zeilennummer ein, so wird die Funktion — vorübergehend — außer Kraft gesetzt, und Sie haben die Möglichkeit z.B. im Direktmodus irgendetwas auszurechnen.

Bei Eingabe einer Zeilennummer mit mindestens einem Zeichen dahinter lebt die Funktion wieder auf. Generell abgeschaltet wird sie mit einfachen AUTO ohne Angabe einer Schrittweite, was in manchen Fällen sicherlich auch sehr hilfreich ist.

4/3.1.1.2

FAST — etwas schneller bitte!
SLOW — normale Geschwindigkeit

Als nächstes sollte man sich entscheiden, mit welcher Geschwindigkeit der Rechner arbeiten soll. Sofern Sie einen 80-Zeichen-Bildschirm verwenden, sollten Sie auf jeden Fall den FAST-Modus einschalten, der Ihren Rechner mit 2 Mhz taktet. Der Rechner wird fast doppelt so schnell.

Wenn Sie ein Fernsehgerät oder einen 40-Zeichen-Monitor besitzen, ist eine Verwendung des FAST-Modus nur sinnvoll, während sich der Computer in einem Rechenprozeß befindet. Wie oft Sie nun in Ihrem Programm zwischen der schnelleren und der langsameren Geschwindigkeit umschalten, bleibt Ihnen überlassen, aber besonders bei längeren Programmen mit rechenintensiven Vorgängen ist die Verwendung des FAST-Befehls anzuraten.

Da der Rechner für jede Monitorart 80/40-Zeichen-Bildschirm einen eigenen Videoprozessor besitzt, aber nur der Videoprozessor für den 80-Zeichen-Bildschirm mit der Geschwindigkeit von 2 MHz zurecht kommt, liegt es auf der Hand, daß Ausgaben auf dem 40-Zeichen-Bildschirm nicht möglich sind. Den Geschwindigkeitsunterschied soll ein kleines Programm verdeutlichen:

```
100 T=TI
110 FOR I=1 TO 500
120 PRINT I,I*I,I*I*I
130 NEXT
140 PRINT (TI-T)/60
```

Wenn Sie dieses kleine Programm laufen lassen, wenn Sie Ihren Rechner gerade eingeschaltet haben, so braucht er — wie er dann selbst anzeigt — 57,0166667 Sekunden. Geben Sie anschließend im Direktmodus den Befehl

```
FAST
```

so wird zunächst — falls angeschlossen — der 40-Zeichen-Monitor erst hell. Wenn Sie nun das Programm erneut mit RUN starten, wird die Geschwindigkeitserhöhung direkt auf dem Bildschirm schon anhand der Ausgabegeschwindigkeit deutlich. Im FAST-Modus braucht der Rechner lediglich noch 30,55 Sekunden.

Sie haben richtig gelesen: Obwohl der Rechner doppelt so schnell getaktet wird, ist der Geschwindigkeitszuwachs nicht mit dem Faktor 2 umschrieben, sondern — in unserem Beispielprogramm — aufgerundet 1,87. Dies hat etwas mit der relativ langsamen Bildschirmausgabe zu tun.

Ein anderes Beispielprogramm soll den Einfluß der Bildschirmausgabe auf die Rechengeschwindigkeit verdeutlichen.

```
100 T=TI
110 FOR I=1 TO 10000
120 A=I*I*I*I
130 NEXT
140 PRINT (TI-T)/60
```

Im SLOW-Modus beträgt die Rechenzeit 116,816667 Sekunden, wohingegen im FAST-Modus nur 55,9833333 Sekunden benötigt werden. Dies entspricht einem Faktor von gerundeten 2,09. Nanu?

Besonders bei rechenzeitintensiven Programmen (Simulationen, statistische Auswertungen, aber auch Sortierverfahren), sollte man auf den FAST-Modus also nicht verzichten.

4/3.1.1.3

PRINT USING — formatiert ausgeben
PUDEF — Ausgabezeichen ändern

Endlich auch bei Commodore: Eine PRINT USING-Anweisung mit der nun auch Tabellen ohne Schwierigkeiten erstellt werden können, ohne daß jedesmal ein eigenes Unterprogramm herangezogen werden muß. Besonders lobenswert ist auch die PUDEF-Anweisung, mit der es sogar sehr einfach möglich ist, Zahlenkolonnen in der deutschen kaufmännischen Zahldarstellung mit Dezimalkomma und Tausenderpunkt darzustellen (siehe auch hierzu das Beispiel im Handbuch auf Seite 4-93).

An dieser Stelle wollen wir nicht näher auf das Format der PRINT USING-Anweisung hinweisen, da dies im Handbuch ausführlich erläutert ist. Lassen Sie sich von den vielen Möglichkeiten zunächst nicht abschrecken, der PRINT USING-Befehl ist einer genauen Prüfung wert. Nicht nur bei der Ausgabe auf dem Bildschirm oder Drucker kann eine Formatierung von Zahlenkolonnen erfolgen, sondern auch bei Ausgabe in eine Datei auf einen Datenträger, wodurch auch geordnete Verhältnisse in den Datensätzen entstehen.

Bei den möglichen Formatierungen ist nicht nur an den Kaufmann als Anwender gedacht, sondern auch an den Wissenschaftler, was sich in den Beispielen bei der Exponentialdarstellung ausdrückt.

4/3.1.1.4

GETKEY — ein Zeichen von der Tastatur einlesen

Was bei anderen Rechnern der INKEY-Befehl darstellt, ist beim C 128 der GETKEY-Befehl. Mit ihm werden umständliche Programmierungen der Form „GET A$:IF A$“ “THEN . . . “ überflüssig, da der GETKEY-Befehl wartet, bis eine Taste gedrückt wurde. Besonders hervorzuheben ist die Möglichkeit, eine Liste von Zeichenkettenvariablen bei diesem Befehl einzugeben. Eine Anwendungsmöglich-

keit ist z.B. das „blinde“ Erfassen eines Paßwortes, wie folgendes kleine Programm zeigt:

```
100 PRINT"BITTE PASSWORT EINGEBEN"
110 GETKEY A$,B$,C$,D$,E$,F$,G$,H$
120 IF A$<>"P" THEN 1000
130 IF B$<>"A" THEN 1000
140 IF C$<>"S" THEN 1000
150 IF D$<>"S" THEN 1000
160 IF E$<>"W" THEN 1000
170 IF F$<>"O" THEN 1000
180 IF G$<>"R" THEN 1000
190 IF H$<>"D" THEN 1000
200 PRINT"SIE DUERFEN WEITERMACHEN"
210 END
220 :
1000 PRINT"IST NICHT
```

Listing 4/3.1.1.4-1

Welchen Aufwand nach der alten Methode! Natürlich kann man die Zeilen ab 120 noch mit der ODER-Verknüpfung zusammenfassen, wenn ein kürzeres Paßwort vorliegt.

4/3.1.1.5

RENUMBER — Ordnung in den Zeilennummern

Vom Programmierer immer wieder gebraucht, jetzt erstmals auch bei einem Commodore-Rechner ab Werk: Der RENUMBER-Befehl. Im Gegensatz zum RENUMBER-Befehl bei Simon's Basic — wie sich die 64er Umsteiger sicherlich noch erinnern — werden hier auch alle Sprünge bei GOTO und GOSUB und ähnlichem mit berücksichtigt. Diese Möglichkeit sollte allerdings nicht dazu verleiten, erst einmal nach dem Motto „Kraut und Rüben“ zu programmieren, aber trotzdem ist eine wesentliche Möglichkeit gegeben, durch Auseinanderschieben der Zeilennummer mittels des RENUMBER-Befehls später noch einige Zeilen in ein bestehendes Programm einzufügen.

Auch fertige und ausgetestete Programme können mit dem RENUMBER-Befehl in ein anschauliches Bild gebracht werden.

Einen kleinen Schönheitsfehler hat der RENUMBER-Befehl allerdings doch noch: Wenn Sie Ordnung in ein Programm bringen wollen, sollten Sie tunlichst von vorne anfangen, da ab der angegebenen Zeilennummer alle Zeilen umnumeriert werden, weil keine letzte Zeilennummer angegeben werden kann. Dies ist ein kleiner Nachteil, den man mit geschickter Vorgehensweise jedoch leicht ausgleichen kann.

4/3.1.1.6

DELETE — überflüssige Zeichen löschen

Eine ähnliche Hilfe, wie der RENUMBER-Befehl, bietet der DELETE-Befehl beim „Aufräumen" im Programm. Nun müssen nicht mehr alle Zeilennummern, die aus einem Programm entfernt werden sollen, von Hand eingegeben werden, dies erledigt alles der DELETE-Befehl.

Man sollte sich jedoch vor Augen halten, daß der DELETE-Befehl genauso wie der RENUMBER-Befehl und einige andere in diesem Kapitel vorgestellten Befehle, eigentlich nicht zum Sprachumfang des Basic gehören, sondern Hilfsmittel bei der Programmeditierung darstellen und somit zu den Systemkommandos (Betriebssystem) gehören und nur im Direktmodus verwendet werden können. Selbständernde Programme können mit diesen Befehlen deshalb nicht geschrieben werden.

Kleine Tricks, wie z.B. das Löschen eines Programmteiles während des gesamten Programmablaufes von Zeilennummern die nicht mehr benötigt werden, um mehr Platz für Daten zu bekommen, sind also nicht möglich. Generell ist auch von diesen Formen der Programmierung abzuraten.

4/3.1.1.7

TRON/TROFF — läuft das Programm auch wie es soll?

Und mit den Befehlen TRON/TROFF kann man die sogenannte TRACE-Funktion ein (ON) und aus (OFF) schalten. Besonders beim Programmtest ist die TRACE-Funktion eine wesentliche Hilfe, und hier insbesondere wieder bei Endlosschleifen.

Die TRACE-Funktion zeigt alle vom Programm abgearbeiteten Zeilennummern am Bildschirm (in eckigen Klammern) an. Leider werden die abgearbeiteten Zeilennummern nicht in einem bestimmten Bildschirmbereich oder nur auf die linke Bildschirmseite begrenzt ausgegeben, sondern fortlaufend über den gesamten Bildschirm. Dies erschwert aus optischen Gründen natürlich die Fehlersuche erheblich.

Sollten Sie also einmal bei einem Testlauf eines neu eingetippten Programmes feststellen, daß dies keine weitere Reaktion zeigt, so können sie mit TRON die TRACE-Funktion einschalten und die Zeilennummern, die abgearbeitet wurden, erscheinen auf dem Bildschirm. Vorher sollte man sich jedoch mit dem Drucker einen Ausdruck des Listings machen und nun anhand des Listings verfolgen, was der Computer macht und was er machen soll. Ohne TRACE-Funktion sind solche Fehler nur durch Einbau von Testzeilen und hier auch nur sehr mühsam herauszufinden.

Da die Zeilennummern sehr schnell auf dem Bildschirm angezeigt werden, sollten Sie auf jeden Fall auf die Taste „NO SCROLL“ zurückgreifen, um den Ablauf verfolgen zu können.

4/3.1.1.8

RESTORE — mitten rein in die DATA’s

Bei Basic-Interpretern ist es mittels DATA-Zeilen und der READ-Anweisung möglich, Daten aus dem Programmtext einzulesen. Bisher war es bei den Commodore-

Rechnern immer der Fall, daß die Daten in der Reihenfolge der READ-Anweisungen vom Programmbeginn an gesucht wurden.

In manchen Fällen ist es jedoch wünschenswert, das gleiche Programm mit verschiedenen Daten laufen zu lassen, oder aus anderen Gründen eine bestimmte DATA-Zeile für den nächsten READ-Befehl festzulegen. Diese Möglichkeit wurde jetzt im Basic 7.0 mit dem RESTORE-Befehl geschaffen.

Der RESTORE-Befehl kann neben seiner normalen Anwendung im Programm auch als Testhilfe herangezogen werden. Sind in einem Programm viele DATA-Zeilen mit Werten für die unterschiedlichsten Variablen oder auch ganzen Feldern vorhanden, so kann es durch ein einziges Element zuviel oder zuwenig innerhalb einer DATA-Zeile zu einem TYPE MISMATCH ERROR kommen, wenn z.B. in eine Zahlvariable ein Text eingelesen werden soll. Bei einem gut strukturierten und dokumentierten Programm ist es dann sinnvoll, vor jeder READ-Anweisung einen RESTORE-Befehl auf die entsprechenden Daten zu machen, die mit dem entsprechenden READ-Befehl eingelesen werden sollen.

Eine weitere Möglichkeit bildet die Anwahl von DATA-Zeilen, die vielleicht aufgrund eines Anwendermenüs ausgesucht werden. Aber auch die Wiederverwendung bereits vorhandener Elemente in DATA-Zeilen ist möglich, wie das Beispiel im Handbuch zeigt. Sollen z.B. von einem bestimmten Datenbestand (z.B. Umsätze), mehrere grafische Darstellungen (Liniendiagramm, Balkendiagramm, Tortendiagramm) nacheinander vorgeführt oder zu verschiedenen Auswertungen herangezogen werden, so ist es ohne die RESTORE-Anweisung unumgänglich, die Daten mehrfach einzugeben.

4/3.1.1.9

TRAP — Fehler des Betriebssystems programmäßig abhandeln RESUME — und wieder zurück ins Hauptprogramm

Eine Funktion, die zwar nicht die Programmierarbeit erleichtert, aber dem Anwender die Möglichkeit gibt, aufgetretene Systemfehler eventuell zu beheben, ist das sogenannte Error-Trapping. Gemeint ist damit nicht eine Behebung von Bedienungsfehlern des Anwenders (was durch Plausibilitätsprüfungen in den meisten Fällen behoben werden kann), sondern eine Programmsequenz, die einen vom Betriebssystem gemeldeten Fehler behandelt.

Dies ist ein wesentlicher Vorteil, da das Programm nun nicht mehr automatisch abgebrochen wird, wenn eine der Fehlermeldungen, die im Handbuch in Kapitel 8.1 beschrieben sind, auftritt.

Bevor die Fehlerbehandlung im Programm jedoch aufgerufen werden kann, muß zunächst eine Zeilennummer (der Beginn der Fehlerbehandlungs-Routine) hinter dem TRAP-Befehl festgelegt sein.

Wir wollen das Error-Trapping an einem kleinen Beispiel besprechen. Auch in den besten Anwenderprogrammen kommt es immer wieder mal vor, daß durch bestimmte Datenkonstellation eine Division durch 0 auftritt, die mathematisch — sowie auch im Rechner — nicht zugelassen ist.

Löschen Sie zunächst ein im Hauptspeicher befindliches Programm mit dem NEW-Befehl und geben Sie anschließend ein

```
110 PRINT 12/A
```

Wenn Sie nun das Programm mit RUN starten, so erhalten Sie die normale Fehlermeldung des Betriebssystems: DIVISION BY ZERO ERROR IN 110.

Bevor wir nun den Fehler abfangen, noch eine Besonderheit, die Ihnen später bei der Programmentwicklung sicherlich sehr hilfreich sein kann. Wenn Sie als nächstes eingeben

```
100 TRAP 1000
```

und das Programm mit RUN erneut starten, so erscheint die Fehlermeldung „UNDEF'D STATEMENT ERROR IN 110". Diesen Tatbestand sollten Sie sich unbedingt merken! Bei komplexeren Programmen kommen Sie leicht in die Lage, nun in Zeile 110 einen Sprungbefehl, ein Unterprogrammaufruf oder ähnliches zu suchen, da diese Fehlermeldung normalerweise auf einen Sprung zu einer nicht vorhandenen Zeilennummer hindeutet. Der beschriebene Fehler liegt allerdings nicht in Zeile 110, sondern in Zeile 100, da ja hier die Zeile 1000 definiert wurde.

Dies ist eine Eigenart der Fehlerbehandlungs-Routine, da sie vom Interpreter erst dann aufgerufen wird, wenn ein Fehler auftritt. Veranschaulichen kann man sich diesen Sachverhalt, wenn man sich nach einem TRAP-Befehl anschließend an jeden Befehl die imaginäre Anweisung „IF Fehler THEN TRAP 1000" vorstellt.

Der TRAP-Befehl ist also nichts anderes als ein spezielles GOSUB, da ein Unterprogramm aufgerufen wird. Statt des RETURN-Befehls muß jedoch am Ende der Fehlerbehandlung ein RESUME-Befehl stehen. Wir ergänzen also die Fehlerbehandlungs-Routine wie folgt:

```
1000 PRINT"ES WURDE DURCH NULL DIVIDIERT.
1010 PRINT"AUTOMATISCHE DIVISORWAHL: 2.5"
1020 A=2.5
1030 PRINT
1040 PRINT EL,ER,ER$(ER)
1050 RESUME
```

Sicherlich ist die Fehlerbehandlungs-Routine jeweils dem Fall anzupassen. Normalerweise werden in der Routine nur Fehler behandelt, die auch vom Programm automatisch erledigt werden können, gegebenenfalls durch Eingabe neuer Daten durch den Anwender. In der Regel wird hier eine Programmverzweigung für alle möglichen Fehler vorgesehen, wobei die Fehlercodes (siehe Handbuch) als Verteiler fungieren können.

Wenn Sie nun das Programm laufen lassen, werden Sie feststellen, daß zusätzlich zu den unterprogrammbedingten Ausgaben noch die Daten ILLEGAL QUANTITY, 65535,-1, und 14 erscheinen. Die in Zeile 1040 bedingte Ausgabe der Zeile, in der der Fehler aufgetreten ist (EL), der Fehlernummer (ER) und dem Fehlertext (ERR$(ER)) bringt hier nicht den Fehler ‚Dividison durch Null'.

Wir haben absichtlich unser Programm nach Zeile 110 nicht beendet, so daß das Programm automatisch in die Fehlerbehandlungs-Routine hineinläuft. Auch an dieser Stelle sollten Sie aufpassen, da hier automatisch ein Fehler erzeugt wird, wenn das Unterprogramm auch ohne Auftreten eines Fehlers angesprungen wird.

Wir ergänzen als noch

```
120 END
```

Wenn Sie jetzt das Programm erneut starten, erscheinen die Texte der Fehlermeldung, sowie die Zeile

```
110    20    DIVISION BY ZERO
```

und das Endergebnis mit dem Wert 4.8. Die Zeile, in der der Fehler aufgetreten ist, wurde als erstes genannt und die anderen Angaben (20/DIVISION BY ZERO) können Sie anhand des Handbuches nachprüfen.

Auf jeden Fall muß der Fehler innerhalb der Routine behoben werden, wenn am Ende der Routine ein RESUME steht und das Programm also an der Stelle fortgesetzt wird, an der der Fehler aufgetreten ist. Achtung: *Nicht hinter* der Stelle, wo der Fehler aufgetreten ist, da sich sonst eine Fehlerbehandlung erübrigen würde. Dies können Sie ausprobieren, indem Sie Zeile 1020 löschen und das Programm erneut starten. Sie erhalten nun fortlaufend die Ausgabe der Fehlermeldungen, was gleichbedeutend mit dem Ausgangszustand (Abbruch des Programmes) ist.

Das Error-Trapping ist eine sehr zweischneidige Sache und sollte nur dann angewendet werden, wenn man sich über die Art von auftretenden Fehlern und deren Behebung im klaren ist. Die Fehlerbehandlung ist insbesondere für Software-Häuser interessant, die bei entsprechender Programmierung dem Anwender eine Hilfestellung zur Behebung des Fehlers geben können.

Im Programm können auch mehrere Fehlerbehandlungs-Routinen vorkommen, die sich auf einzelne Fehler spezialisieren. Infolge des Programmablaufes ist normalerweise klar, welcher Fehler unter Umständen an welcher Stelle auftreten kann. Vor eine solche unsichere Programmeinweisung sollte man den TRAP-Befehl mit der entsprechenden Zeilennummer für den Fehler stellen.

Computerneulingen sei von der Verwendung des Error-Trappings abgeraten. Wie bereits erwähnt, sind intensive Kenntnisse vonnöten, wann wo welcher Fehler in welcher Form auftreten kann.

Eine andere kleine Hilfestellung bei Auftreten eines Fehlers wollen wir im nächsten Unterkapitel behandeln.

4/3.1.1.10

HELP — Hilfe, wo ist der Fehler?

Ist trotz verwendeter TRAP-Funktion das Programm wegen eines Fehlers abgebrochen worden, so kann man sich die fehlerhafte Zeile mit dem HELP-Befehl (auch erreichbar durch Drücken der HELP-Taste) ansehen. Dies erspart jedoch nur ein „LIST-Zeilennummer“, da ja die fehlerhafte Zeile bei einem Programmabbruch angegeben wird.

Der HELP-Befehl bietet aber noch eine weitere — wenn auch kleine — Hilfe. Die fehlerhafte Anweisung wird unterstrichen (80-Zeichen-Bildschirm) oder in inverser Schrift dargestellt (40-Zeichen-Bildschirm). Dies ist natürlich nicht sehr hilfreich, wenn nur eine einzige Anweisung in einer Zeile steht. Außerdem werden alle Anweisungen nach der fehlerhaften Anweisung ebenfalls mitunterstrichen, so daß in einer Programmzeile mit eventuell zehn Anweisungen und aufgetretenen Fehlern in der ersten Anweisung die gesamte Programmzeile unterstrichen ist — was natürlich keine große Hilfe ist. Vorteilhaft ist die HELP-Funktion also nur, wenn der Fehler

irgendwo miten in einer Programmzeile auftritt. Veranschaulichen können Sie sich den Sachverhalt indem Sie eingeben

```
100 PRINT 12/A : PRINT A
```

Es erscheint der im letzten Kapitel schon erwähnte DIVISION BY ZERO ERROR IN 100. Drücken Sie nun die HELP-Taste, so wird die gesamte Zeile unterstrichen. Wenn Sie beide Anweisungen umdrehen, also schreiben

```
100 PRINT A : PRINT 12/A
```

so erscheint nach einem Programmstart und Drücken der HELP-Taste nach dem Programmabbruch die Zeile 100 auf dem Bildschirm, jedoch ist nur die zweite Anweisung unterstrichen.

4/3.1.1.11

Strukturbefehle

Neben den immer vorhandenen Strukturvariablen IF . . . THEN, GOTO und GOSBUB . . . RETURN, sowie FOR . . . NEXT, bietet das Basic 7.0 des C 128 noch ein paar weitere Möglichkeiten an.

4/3.1.1.11.1

IF . . . THEN . . . ELSE — entweder . . . oder

Neben einigen anderen Befehlen wurde auch die Befehlskonstellation für bedingte Verzweigung verbessert. Umständliche Programmsprünge, wenn bei einer bedingten

Verzweigung sowohl der „wenn dann“-Teil als auch der „wenn nicht, dann“-Teil eine Programmfunktion erfüllen sollten. Passen die Anweisungen für die beiden Aktionsteile nicht mehr in eine Zeile, so hilft die im nächsten Unterkapitel beschriebene Blockkonstruktion.

4/3.1.1.11.2

BEGIN/BEND — Blöcke in Basic

Blöcke und doch keine Blöcke. Wer Programmblöcke aus den sogenannten blockorientierten Sprachen (z.B. PL/1 oder Pascal) gewohnt ist, wird von den Blöcken beim C 128 etwas enttäuscht sein. Hier ist nicht die Rede von strukturierter Programmierung oder lokalen und globalen Variablen, sondern nur einer speziellen Einsatzform von Blöcken, die es auch bei den höheren Programmiersprachen gibt. Globale und lokale Variablen sind bei Basic-Interpretern sowieso nur mit Tricks und Kniffen möglich.

Die BEGIN- und BEND-Anweisungen sind jedoch ein entscheidender Vorteil, da sie andere Unzulänglichkeiten der Sprache Basic im Allgemeinen überdecken. Im letzten Kapitel haben wir die erweiterte IF . . . THEN . . . ELSE . . .-Funktion besprochen. In manchen Fällen reicht jedoch der Platz innerhalb einer Zeile nicht aus, wenn sowohl die Bedingung als auch der Anweisungsteil, wenn die Bedingung erfüllt ist, als auch der Anweisungsteil, wenn die Bedingung nicht erfüllt ist, innerhalb in einer Zeile stehen müssen. Besonders eng wird es innerhalb einer Zeile, wenn komplexe Bedingungen zugrunde liegen, die mit mehreren UND- und/oder ODER-Verknüpfungen verknüpft sind. Hier schafft der Block mit BEGIN(n) und BEND (**B**lock **END**e) Abhilfe, da bedingte Anweisungen nun über mehrere Zeilen gestreckt werden können.

4/3.1.1.11.3

DO — neuartige Schleifen

Die DO-Anweisung ist ein ähnlicher Strukturbefehl, wie dies für Programmschleifen mit FOR . . . NEXT allgemein verwendet wird, weist jedoch einige Besonderheiten auf. So ist es z.B. in manchen Fällen nötig, eine Schleife unendlich oft laufen zu lassen — jedenfalls per Programmierung — bis ein bestimmtes Kriterium vom Anwender eingegeben wird. Bei FOR . . . NEXT-Schleifen mußte man sich mit der Konstruktion

```
FOR I=0 TO 1E38
```

behelfen, da durch interne Programmierung bedingt, das Schleifenendekriterium nie vom Rechner erkannt wird. Auch die Konstruktion mit GOTO „Anfang“ wurde in einem solchen Fall sehr häufig verwendet, wenn es auch nicht unbedingt den Regeln der strukturierten Programmierung entspricht.

Dies kann man nun durch einfache Verwendung von DO erreichen. Dem NEXT bei den bisher gebräuchlichen Programmschleifen entspricht dabei das LOOP, wobei jedoch hier auch die Schleife mit einem gesonderten Befehl (EXIT) verlassen werden kann. Dies ist eine weitere Besonderheit, da das Verlassen von FOR . .. NEXT-Schleifen keine besonders elegante Programmierung darstellt.

Durch die Bedingung innerhalb der Programmschleife mit DO (WHILE) ist eine weitere Besonderheit gegeben, die in manchen Fällen sehr sinnvoll ist, und umständliche IF-Abfragen innerhalb einer Programmschleife überflüssig macht. Als Beispiel seien hier nur mathematische Näherungsverfahren erwähnt. Eine iterative Berechnung wird sooft durchgeführt, bis das Ergebnis der gewünschten Genauigkeit entspricht. Dabei sollten die Ergebnisse von zwei aufeinanderfolgenden Durchläufen der Iteration in zwei separaten Variablen — die meist sowieso gebraucht werden — abgelegt werden. Ein Abbruchkriterium für eine DO-Schleife könnte dann lauten:

```
DO WHILE (WERTALT — WERTNEU < 0.000001)
```

4/3.1.4

Verwalten von Daten und Programmen mit externen Speichermedien

Stehen im 64er-Modus nur relativ wenige Befehle (OPEN, LOAD, SAVE, CLOSE und VERIFY) für die Arbeit mit externen Speichermedien zur Verfügung, so bietet der C 128-Modus eine ganze Palette, die hauptsächlich im Hinblick auf die Diskettenstation abgestimmt sind.

4/3.1.4.1

HEADER — Die Formatierung

Bevor Sie die Arbeit mit einer Diskettenstation aufnehmen können, müssen Sie die Disketten zur Aufnahme von Daten vorbereiten. Dieser Vorgang nennt sich Formatieren und wird von dem HEADER-Befehl unterstützt, so wie auch das Löschen des Inhaltes auf einer nicht mehr benötigten Diskette mit Hilfe des HEADER-Befehles erledigt werden kann.

Da es sinnvoll ist, alle Disketten gleich nach dem Kauf zu formatieren, um späteren Komplikationen aus dem Weg zu gehen, sollte man die HEADER-Anweisung in ein kleines Programm einbauen. Dieses Programm kann aus einer Endlosschleife bestehen, in der nach jedem HEADER-Befehl ein Wartebefehl für einen Tastendruck angegeben ist.

```
100 DO
110 INPUT"NAME DES DISKETTE";NA$
120 IF LEN(NA$)> 16 THEN 110
130 IF NA$="ENDE" THEN END
140 INPUT"LAUFWERK              ";L
150 IF L<> 0 AND L<>1 THEN 140
160 INPUT"KENNUNG               ";K$
170 IF LEN(KE$)> 2 THEN 160
180 INPUT"GERAET                ";G
190 IF G<4 OR G>15 THEN 180
200 HEADER (NA$),IP1,D(L),U(G)
210 LOOP
```

Listing 4/3.1.4.1

Bei dem uns zur Verfügung stehenden Rechner stellte sich jedoch heraus, daß bei der Kennung keine Variablen möglich waren, da sonst ein SYNTAX ERROR entstand.

4/3.1.4.2

CATALOG/DIRECTORY — Das Inhaltsverzeichnis

Nachdem Sie eine Diskette formatiert haben, aber auch zu jedem anderen Zeitpunkt, können Sie sich das Inhaltsverzeichnis einer Diskette mit den Befehlen CATALOG und DIRECTORY anzeigen lassen. Bei einer formatierten Diskette erscheint als Überschrift neben der Laufwerksnummer der vorher gewählte Name, dann die Kennung und ein '2A', das jedoch für den Anwender bedeutungslos ist. In Abhängigkeit von der verwendeten Floppy-Station (1541 oder 1571) wird die Anzahl der freien Blocks (je 256 Byte) angezeigt. Die Anzeige des Inhaltsverzeichnisses zerstört im 128er-Modus nicht das im Hauptspeicher befindliche Programm (im 64er-Modus muß das Inhaltsverzeichnis als Programmdatei geladen werden).

Befinden sich Dateien auf der Diskette, so wird neben den Namen auch die Anzahl der verwendeten Blocks und die Art der Datei geladen. PRG bedeutet hierbei Programmdatei, worunter auch binäre Dateien geführt werden, wie z.B. die Daten für Sprites. Weiterhin wird bei Datendateien zwischen sequentiell (SEQ) und relativen Dateien/Direktzugriffsdateien (REL) unterschieden.

Da diese beiden Befehle häufig verwendet werden, sollen an dieser Stelle noch die Abkürzungen erwähnt werden: CATALOG wird mit C SHIFT-A aufgerufen und DIRECTORY mit DI SHIFT-R. Sparkünstler wählen also die Abkürzung für CATALOG.

Bei großen Inhaltsverzeichnissen sollte man auch auf die sogenannte Präfixsuche (mit » * «) bzw. Suche mit Platzhaltern (»?«) zurückgreifen.

4/3.1.4.3

BACKUP — Die Datensicherung

Leider funktioniert der BACKUP-Befehl nur bei zwei angeschlossenen Laufwerken (nicht unbedingt ein Doppellaufwerk). Da er ausreichend im Handbuch beschrieben wurde, soll an dieser Stelle nur etwas zum Thema Datensicherung allgemein gesagt werden.

Eingebürgert hat sich eine sogenannte Großvater-Vater-Sohn-Datensicherung. Dies bedeutet, daß die tägliche Datensicherung nicht immer auf die gleiche Diskette gemacht wird, sondern für jede Originaldiskette zwei BACKUP-Disketten benötigt werden. Da alle drei Disketten im Prinzip sowohl Original- als auch Kopiedisketten sind, wollen wir sie im folgenden kurz mit A, B und C bezeichnen.

Bei der ersten Sicherung wird die Originaldiskette A auf die Diskette B kopiert. Bei den weiteren Arbeiten wird Diskette B benutzt, um beim nächsten Sicherungsvorgang auf C kopiert zu werden. Somit stellt zunächst A die Kopiediskette des Originals B dar und anschließend B die Kopiediskette des Originals C. Der nächste Kopievorgang erfolgt von der Diskette C auf A, die somit wieder Originaldiskette wird. Dieser Vorgang wiederholt sich fortlaufend, wobei jedoch die Abnutzung der Disketten zu berücksichtigen ist. Eine zeitliche Aussage kann generell hier nicht getroffen werden, da dies von der unterschiedlichen Benutzungshäufigkeit der Disketten abhängt.

Sollte an der Diskette C etwas geschehen, so beinhaltet die Diskette B den letzten Datenstand und die Diskette A zwar einen weniger aktuellen Datenstand, was jedoch besser als vollständig verlorene Daten ist. Es ist bei keinem Computer auszuschließen, daß beim BACKUP eine Diskette kaputtgeht. Damit nun nicht alle Daten neu erfaßt werden müssen, dient die Kopiediskette der zweiten Generation als zusätzliche Sicherheit.

4/3.1.4.4

SCRATCH — Das Löschen

Damit nicht alle Daten bis in Ewigkeit auf den Disketten verbleiben, kann der SCRATCH-Befehl zum Löschen von Dateien angewendet werden. Dabei ist es unerheblich, ob eine Programmdatei oder eine Datendatei (sequentiell oder relativ) gelöscht werden soll.

Bei der Verwendung des SCRATCH-Befehls ist größte Vorsicht geboten, da gelöschte Dateien nur mit sehr großem Aufwand und nur von eingefleischten Profis wiederhergestellt werden können (1571-Besitzer haben es besser: auf der Systemdiskette befindet sich ein entsprechendes Hilfsprogramm).

Beim Löschen einer Datei wird im Inhaltsverzeichnis der Diskette ein Merker gesetzt, der die Datei als gelöscht kennzeichnet. Außerdem werden die benötigten Blocks wieder als frei für weitere Dateien gekennzeichnet. Auch mit einem Hilfsprogramm ist eine Reparatur meist nur möglich, wenn keine Datei seit dem Löschen neu angelegt oder vergrößert wurde.

Anders als bei den Rechnern der 8000er-Serie wird der SCRATCH-Befehl beim C 128 mit SC SHIFT-R abgekürzt.

4/3.1.4.5

RENAME — Die Namensänderung

Vielleicht ist es Ihnen schon beim Speichern einer Programmdatei passiert, daß die Fehlermeldung „FILE EXISTS ERROR“ erschienen ist. Wenn Sie nun die alte Datei nicht löschen wollen und außerdem die neue Datei den speziell vorgegebenen Namen erhalten soll, so kann der RENAME-Befehl Ihnen wertvolle Dienste leisten.

Die Datei mit dem veränderten Namen weist außer dieser Änderung keine weiteren Differenzen auf. Lediglich der Dateiname im Inhaltsverzeichnis der Diskette wird

geändert. Es geschieht also kein Umkopieren der Datei mit einem anderen Namen, was die Angabe eines einzigen Floppylaufwerks beinhaltet. Dies im Gegensatz zum COPY-Befehl, mit dem man Dateien — auch mit veränderten Namen und zwischen verschiedenen Laufwerken — umkopieren kann.

4/3.1.4.6

COPY — Das Kopieren

Der COPY-Befehl ermöglicht das Umkopieren einzelner Dateien zwischen verschiedenen Laufwerken, aber auch innerhalb der gleichen Diskette. Dabei kann der Name beibehalten (wenn zwei verschiedene Laufwerke benutzt werden) oder auch geändert werden.

Die Verwendung des COPY-Befehls ist immer dann angezeigt, wenn **einzelne** Dateien gesichert oder aus anderen Gründen auf eine andere Diskette überspielt werden müssen. Beim Überspielen auf eine andere Diskette muß unbedingt ein zweites Laufwerk vorhanden sein, da eine Zwischenspeicherung der Daten (Programmdateien, sequentielle Dateien, relative Dateien) innerhalb des Hauptspeichers des C 128 nicht erfolgt.

Sehr wichtig ist der COPY-Befehl, wenn Sie bei Anwendung des BACKUP-Befehls auf einen READ ERROR oder WRITE ERROR gelaufen sind, und somit das komplette Duplizieren einer Diskette nicht möglich ist. Wenn Sie Glück haben, befindet sich der Schreib-/Lesefehler an einer Stelle, an der die Diskette nicht mit Ihren Daten beschrieben ist. Da der BACKUP-Befehl jedoch alles kopiert, bricht er bei einem Schreib-/Lesefehler ab.

In diesem Fall können Sie mit dem COPY-Befehl die Dateien einzeln kopieren, was aus Gründen der Datensicherheit sehr zu begrüßen ist.

4/3.1.4.7

COLLECT — Die Garbage Collection — Aufräumungsarbeiten

Der COLLECT-Befehl ist nur interessant, wenn sich aus irgendwelchen Gründen eine nicht geschlossene Datei auf der Diskette befindet. Solche Dateien werden mit einem Sternchen vor der Dateiart (PRG, SEQ, REL) angezeigt. Normalerweise sind solche Dateien nicht mehr zu retten, und man sollte insbesondere bei sequentiellen Dateien darauf achtgeben, daß sie nach Programmende auch geschlossen werden.

Nicht immer können solche mit » * « gekennzeichnete Dateien mit dem SCRATCH-Befehl gelöscht werden. Hier kann man nun den COLLECT-Befehl anwenden. Dies sollte allerdings nur im äußersten Notfall passieren, wenn die Datei gar nicht mehr anders gerettet werden kann, da sie dann unwiederbringlich verloren ist.

4/3.1.4.8

DCLEAR — Die Floppy-Kanäle schließen

Der DCLEAR-Befehl ist kein Hilfsmittel, die im letzen Kapitel erwähnten ungeöffneten Dateien zu schließen. Deshalb sollten alle Dateien vor Verwendung des DCLEAR-Befehls geschlossen sein.

Es stehen dem Anwender nur eine begrenzte Zahl von Kanälen zur Datenübergabe/Übernahme von/zur Floppy zur Verfügung. Durch unsaubere Programmierung (DCLOSE vergessen) oder durch andere Einwirkungen (Programmabbruch, manchmal auch bei Diskettenwechsel) kann es vorkommen, daß die Floppykanäle mit dem DCLOSE-Befehl nicht mehr geschlossen werden können. In diesem — seltenen — Fall leistet der DCLEAR-Befehl wertvolle Hilfe.

4/3.1.4.9

DVERIFY — VERIFY für Disketten

Der selten gebrauchte Befehl DVERIFY bildet das Gegenstück zum VERIFY-Befehl. Beim Kassettenrekorder ist der VERIFY-Befehl unbedingt nach dem Speichern eines Programmes anzuwenden, um das korrekte Speichern der Programmdaten zu überprüfen. Um nun auch eine Überprüfung bei einer Diskettenstation möglich zu machen, wurde der Befehl DVERIFY kreiert. Sie können also nach dem Speichern eines Programmes mit DSAVE überprüfen, ob der Vorgang korrekt verlaufen ist.

Dieser ist jedoch nicht so wichtig, jedenfalls nicht für die Prüfung gespeicherter Programme sofort nach dem Speichern. Dies wird vom Betriebssystem automatisch erledigt, d.h. nach dem Schreiben wird automatisch ein Prüflauf durchgeführt.

Der DVERIFY-Befehl kann aber an anderen Stellen sinnvoll eingesetzt werden, wenn man z.B. nicht weiß, ob man eine neue Programmversion bereits auf Diskette gespeichert hat oder nicht. Hier wird der DVERIFY-Befehl also nicht zum Prüfen gespeicherter Programme eingesetzt, sondern zur Unterschiedsfindung. Der DVERIFY-Befehl bezieht sich nur auf Programm- und nicht auf Datendateien.

4/3.1.4.10

BLOAD/BSAVE — Speicherteile ein- und auslagern

Besonders Maschinenspracheprogrammierer werden die Befehle BLOAD und BSAVE sehr begrüßen, da sie das Speichern Ihrer Maschinenprogramme wesentlich vereinfachen. Aber auch Spritedaten können auf diese einfache Weise gespeichert und später an die richtige Stelle geladen werden, wie im Kapitel Grafik mehrfach aufgezeigt ist. Eine weitere Möglichkeit bildet das Speichern von Grafiken, die ja ebenfalls im RAM dargestellt sind.

Im 64er-Modus ist lediglich das Laden von binären Dateien an eine, in der Datei vorgesehenen, Stelle möglich, indem man beim LOAD-Befehl hinter der Geräte-

nummer noch ein »,1« schreibt. Das Gegenstück vom BSAVE-Befehl ist dort nicht vorhanden.

Aber auch komplexere Anwendungen, wie z.B. das Auslagern ganzer Bänke auf Diskette, und späteres wieder Einladen, was besonders bei großen Programmsystemen erforderlich sein kann, wird durch die Befehle BLOAD und BSAVE unterstützt.

4/3.1.4.11

DLOAD/DSAVE — Programme laden und speichern

Ähnlich den Befehlen BLOAD und BSAVE fungieren die Befehle DLOAD und DSAVE, die sich jedoch auf normale Programmdateien beziehen. Sie ersetzen die Befehle LOAD und SAVE vom 64er bzw. 64er-Modus, und ersparen lediglich die Angabe von »,8« hinter dem Dateinamen.

Da diese Befehle normalerweise recht häufig verwendet werden, sind ihnen Funktionstasten zugeteilt, »DLOAD"« erhalten Sie durch Drücken der Funktionstaste F2 und »DSAVE"« durch F5. Nach Drücken der Funktionstasten brauchen Sie also lediglich die Dateinamen einzugeben und die RETURN-Taste zu drücken.

Abgekürzt werden die Befehle mit D SHIFT-L und D SHIFT-S, was ebenfalls die Handhabung vereinfacht.

Eine weitere Besonderheit bietet die Tastenkombination RUN/STOP - SHIFT. Drücken Sie diese Tastenkombination, so erscheint auf dem Bildschirm die Abkürzung für DLOAD, ein Anführungszeichen gefolgt von einem Stern, mit dem das erste Programm von der Diskette geladen **und** sofort gestartet wird. Dies vereinfacht besonders Computerlaien die Handhabung ihres Rechners, wenn sie nicht mit ihm programmieren, sondern ihn nur anwenden wollen.

4/3.1.4.12

BOOT — Das Starten einer Binärdatei

Ähnliches was mit DLOAD"* für normale Programmdateien gilt, kann mit dem BOOT-Befehl für Binärdateien realisiert werden. Es ist offensichtlich, daß es sich hierbei um Maschinenprogramme handeln muß, die vorher mit BSAVE-Befehl gespeichert worden sind.

Das Laden einer Binärdatei lediglich durch Angabe von BOOT bedarf schon etwas eingehenderer Kenntnisse, da der Sektor 0 in Spur 1 der Diskette nicht mit dem BSAVE-Befehl beschrieben werden kann. Hierzu muß — neben anderen Befehlen — der BLOCK-WRITE-Befehl herangezogen werden.

An dieser Stelle soll das Handbuch der Commodore-Floppy 1571 einmal angeführt werden, da es neben einer genaueren Beschreibung der Floppymöglichkeiten sowohl in bezug auf Basic 2.0 als auch Basic 7.0 auch auf weitere Feinheiten eingeht. Mit den entsprechenden Hilfsprogrammen wird u.a. das Kopieren einer Diskette mit einem Einzellaufwerk, aber auch das Rückgängigmachen des SCRATCH-Befehls beschrieben. Dieses Handbuch ist sicherlich auch für 1541-Besitzer interessant, da vieles auch für diese Floppy gilt.

4/3.1.4.13

DS und DS$ — Die Fehlermeldung

Hervorzuheben ist auch die Eigenschaft des C 128, die Fehlermeldungen von der Floppy sehr einfach abzufragen. Hier fungieren die beiden Systemvariablen DS und DS$, wovon DS$ im Direktmodus und als Fehlerausgabe bei Programmabbruch verwendet werden sollte, da der Wert von DS darin enthalten ist.

Die Systemvariable DS beinhaltet die Fehlernummer und gibt dem Programmierer die Möglichkeit, gewollte und reguläre Fehler per Programm abzufangen. So beinhaltet z.B. die Fehlernummer 50 einen RECORD NOT PRESENT ERROR, der

u.a. dann auftritt, wenn ein nicht vorhandener Datensatz aus einer Direktzugriffsdatei gelesen werden soll. Hier kann nach dem Lesen eines Datensatzes eine Abfrage im Programm erfolgen, die ungefähr so aussehen könnte:

```
IF DS=50 THEN PRINT "DATENSATZ NICHT VORHANDEN" : GOTO "WEITER"
```

Generell ist zu beachten, daß es zwei grundsätzliche Typen von Fehlermeldungen gibt, und zwar solche, die das Betriebssystem des C 128 meldet und solche, die von der Floppy an den Rechner übergeben werden.

Generell ist anzuraten, nach jedem Floppy-Befehl eine Abfrage auf DS anzufügen, und das Programm nur bei dem Wert 0 (kein Fehler) weiterlaufen zu lassen. Ist keine ergänzende Fehlerbehandlung gegeben, wie oben bei Fehlernummer 50 dargestellt, so sollte das Program mit

```
PRINT DS$ : END
```

abgebrochen werden.

Damit verhindern Sie Unstimmigkeiten, die sich sonst sehr leicht einschleichen. Insbesondere ist das Vorgehen bei Direktzugriffsdateien (relativen Dateien) interessant, da diese naturgemäß ständig durch neue Datensätze vergrößert werden. Die Fehlermeldungen FILE TOO LARGE und DISK FULL werden auf diese Weise erkannt. Andernfalls müßte man bei jeder Floppy-Operation ein Auge auf die Leuchtanzeige werfen.

Auf jeden Fall sollten Sie sich bei einer roten Anzeige am Floppylaufwerk die Fehlermeldung auf den Bildschirm holen, um entsprechend reagieren zu können.

4/3.1.4.14

DOPEN — Das Öffnen von Dateien

Nachdem wir bisher die allgemeinen Befehle zum Diskettenhandling besprochen haben, wollen wir uns im Rest von Kapitel 4/3.1.4 der Dateibehandlung im besonderen widmen.

Bevor man mit einer Datei auf der Diskette arbeiten kann, muß zunächst ein Kanal eröffnet werden, über den die Daten transferiert werden können. Dies geschieht mit dem DOPEN-Befehl, der außerdem noch festlegt, welche Datei angesprochen werden soll (Dateiname).

Sequentielle Dateien müssen entweder zum Lesen bzw. Schreiben der Daten geöffnet werden. Beim Schreiben ist an den Dateiname ein »,W« anzufügen, das Lesen benötigt keinen weiteren Parameter.

Anders als bei den Sound- und Grafikbefehlen braucht beim DOPEN-Befehl kein Komma gesetzt zu werden, wenn ein Parameter weggelassen wird. Die Zuordnung der einzelnen Parameter zu ihrer Bedeutung wird durch die Buchstaben L (Länge bei relativen Dateien), D (Laufwerk), U (Geräteadresse) und W (Schreiben bei sequentiellen Dateien) festgelegt.

Eine Datei mit dem gleichen Dateinamen kann nicht nochmals zum Schreiben für eine sequentielle Datei geöffnet werden, sonst erscheint die Fehlermeldung 'FILE EXISTS ERROR'. Sollte eine solche Datei trotzdem noch mal beschrieben werden müssen, so ist sie vorher mit dem SCRATCH-Befehl zu löschen.

Bei relativen Dateien ist es nicht nötig, diese entweder zum Schreiben oder zum Lesen zu öffnen, bei wiederholtem Öffnen der Datei braucht auch nicht mehr die Länge mit angegeben zu werden. Diese ist nur beim Anlegen der Datei wichtig.

An dieser Stelle noch etwas zu den Begriffen sequentielle Datei und relative Datei. Sequentielle Dateien werden so bezeichnet, weil die Daten nacheinander (sequentiell) in diese Datei eingetragen werden. Solche Dateien lassen sich nur als Ganzes beschreiben oder lesen. Einzelne Daten können nicht aus der Datei entnommen bzw. hineingeschrieben werden. Das Entnehmen einzelner Daten kann jedoch durch Überlesen aller vorhergehenden Daten in der sequentiellen Datei einfach realisiert werden. Ein gezieltes Hineinschreiben in die Datei wird jedoch durch den Aufwand nicht gerechtfertigt. Vorteile bieten sequentielle Dateien vor allem bei Daten, die auch sequentiell verarbeitet werden, wie z.B. statistische Auswertungen.

Relative Dateien bieten gegenüber sequentiellen Dateien einen entscheidenden Vorteil, indem auf einzelne Datensätze zugegriffen werden kann. Wenn eine relative (Direktzugriffs-)Datei geöffnet wurde, so kann sowohl gelesen als auch geschrieben werden. Es ist also nicht nötig, vor jedem schreibenden Zugriff ein »,W« bei einem DOPEN-Befehl anzuhängen und die Datei nach dem Schreiben zu schließen.

Besonders in der kommerziellen Datenverwaltung finden heute Direktzugriffsdateien allgemeine Anwendung. Sie gestatten es z.B., die Daten eines einzelnen Kunden gezielt aus der Datei herauszuholen, diese zu ändern und gezielt wieder zu speichern. Wir werden diese Vorgehensweise später noch etwas näher beleuchten (siehe auch Kapitel 4/4.7.).

Wie nach jedem Floppy-Befehl sollte auch nach DOPEN — und hier besonders — eine Abfrage der Fehlervariablen DS nicht fehlen.

4/3.1.4.15

DCLOSE — Das Schließen von Dateien

Das Gegenstück zum DOPEN-Befehl bildet der DCLOSE-Befehl. Er bezieht sich jeweils auf die geöffnete Datei, sofern diese hinter einem '#' angegeben bzw. auf alle geöffneten Dateien. Vor einem Programmende sollte möglichst immer der Befehl DCLOSE gegeben werden, um besonders bei sequentiellen Dateien zu vermeiden, daß die Dateien nachher mit dem COLLECT-Befehl geschlossen werden müssen, was den Datenverlust zur Folge hat.

Werden mehrere Dateien — insbesondere relative Dateien — in einem Programm benötigt, so werden Sie sehr schnell auf die Fehlermeldung NO CHANNEL stoßen. Dies bedeutet, daß für das Öffnen einer weiteren Datei kein Kanal mehr freigegeben werden kann. Die Anzahl der Kanäle ist begrenzt, wobei jede relative Datei drei Kanäle benötigt und jede sequentielle Datei einen.

In diesem Fall sollte man in Kenntnis des Programmlaufes eine zur Zeit nicht benötigte Datei schließen, um sie später — wenn eine andere nicht mehr benötigte Datei geschlossen wurde — wieder mit dem DOPEN-Befehl zu öffnen.

Im folgenden zeigten wir Ihnen ein kleines Programm, mit dem die ersten 100 Zahlen in die Datei TESTDATEI 1 geschrieben werden.

```
100 DOPEN #2,"TESTDATEI1",W
110 IF DS <> 0 THEN PRINTDS$ : END
120 FOR I=1 TO 100
130 PRINT#2,I
140 PRINT I
150 NEXT
160 DCLOSE #2
170 IF DS <> 0 THEN PRINTDS$ : END
```

Listing 4/3.1.4.15-1

Wenn Sie das Programm laufen lassen, werden Sie feststellen, daß das Öffnen der Datei eine Zeitlang benötigt. Außerdem wird ca. eine Sekunde für das Schließen der Datei benötigt, und wenn Sie aufmerksam beobachten, wird während des Speicherns ebenfalls noch eine kleine Pause (am Bildschirm) eingelegt, in der die Floppy den nächsten Block anspricht, da nicht alle Zahlen innerhalb eines Blockes Platz finden.

An unserem Programmbeispiel wird auch gleich die Benutzung des PRINT-Befehls deutlich, wenn dieser nicht auf den Bildschirm, sondern auf Diskette geschrieben werden soll.

Wenn Sie nun das Programm ein zweitesmal starten, laufen Sie auf den Fehler

```
63, FILE EXISTS,00,00
```

was beim DOPEN-Befehl ein Überschreiben bereits vorhandener Dateien verhindert.

Ändern Sie nun in Zeile 100 den Dateinamen auf TESTDATEI2 und starten Sie das Programm erneut. Sie haben nun zwei Testdateien, die jeweils die Zahlen 1 bis 100 beinhalten.

Diese Daten können wir nun mit einem ähnlichen Programm einlesen, wobei wir den PRINT-#-Befehl durch einen INPUT#-Befehl ersetzen und natürlich den Parameter »,W« weglassen.

An dieser Stelle noch etwas zum INPUT-Befehl: Mit dem INPUT-Befehl können die Daten zwar eingelesen werden, jedoch werden keine Semikolons, Doppelpunkte und Kommata mit eingelesen. Diese werden vom Rechner als Trennzeichen zwischen den Daten interpretiert. Wenn Sie in Ihrer Datei die ebengenannten Zeichen verwendet haben, so müssen Sie den GET#-Befehl verwenden, und damit jedes Zeichen einzeln einlesen, was natürlich eine langwierige Sache ist.

Hier nun das leicht geänderte Listing:

```
100 DOPEN #2,"TESTDATEI1"
110 IF DS <> 0 THEN PRINTDS$ : END
120 FOR I=1 TO 100
130 INPUT#2,I
140 PRINT I
150 NEXT
160 DCLOSE #2
170 IF DS <> 0 THEN PRINTDS$ : END
```

Listing 4/3.1.4.15-2

Generell ist es auch möglich, beim Schreiben Zeichenreihen und Zahlvariablen (Integer und Floating Point) zu mischen, jedoch muß beim Auslesen der Datei die Reihenfolge der Variablentypen die gleiche sein, wie beim Einschreiben.

In unserem Fall wissen wir natürlich, wie viele Einträge (Daten) in der sequentiellen Datei vorhanden sind. Es gibt zwei Methoden, das Dateiende festzustellen. Bei der ersten Methode schreibt man als erstes Datum in die Datei die Anzahl der Datensätze (einzelne Daten oder Datengruppen mit gleichem Schema) und liest dann im

Programmablauf zunächst aus der Datei dieses erste Datum ein, um in eine FOR . . . NEXT-Schleife — die als Schleifenendekriterium diese Zahl enthält — den Rest der Datei einzulesen.

Die andere Möglichkeit macht sich die Statusvariable ST zunutze. Dazu unser nächstes Beispiel, indem wir Zeile 120 gelöscht und Zeile 150 in 'GOTO 130' geändert haben. Außerdem geben wir bei der Ausgabe am Bildschirm noch den Wert der Statusvariablen ST an.

```
100 DOPEN #2,"TESTDATEI1"
110 IF DS <> 0 THEN PRINTDS$ : END
130 INPUT#2,I
140 PRINT I,ST
150 GOTO 130
160 DCLOSE #2
170 IF DS <> 0 THEN PRINTDS$ : END
```

Listing 4/3.1.4.15-3

Wenn Sie dieses Programm starten, so erscheint neben der eingelesenen Ziffer bis zur Zahl 99 als Wert der Statusvariablen eine 0. Bei der ersten 100 (die noch zu unserer Datei gehört) weist der Status einen Wert von 64 auf und alle folgenden Ausgaben geben den zuletzt eingelesenen Wert (100) und den Statuswert 66 wieder. Das Programm kann nur mit der STOP-Taste angehalten werden.

Ist nach dem Einlesen eines Datums das Dateiende erreicht, so meldet dies also die Statusvariable mit dem Wert 64, was wir uns natürlich in einem Programm zunutze machen können.

```
100 DOPEN #2,"TESTDATEI1"
110 IF DS <> 0 THEN PRINTDS$ : END
120 DO
130 INPUT#2,I
140 PRINT I,ST
145 IF ST=64 GOTO 160
150 LOOP
160 DCLOSE #2
170 IF DS <> 0 THEN PRINTDS$ : END
```

Listing 4/3.1.4.15-4

Da man es als unsaubere Programmierung bezeichnen kann, wenn man aus einer FOR . . . NEXT-Schleife herausspringt, verwenden wir in unserem Fall die Konstruktion DO . . . LOOP. Diese setzen wir wieder in die Zeilen 120 und 150 ein, und ergänzen Zeile 145 mit dem Aussprung, wenn die Datei zu Ende ist.

4/3.1.4.16

APPEND — Daten an Datei anhängen

Der APPEND-Befehl ist eine Ergänzung zum DOPEN-Befehl in einem speziellen Fall. Um die Handhabung der sequentiellen Dateien etwas zu vereinfachen, kann der APPEND-Befehl herangezogen werden. So ist es immerhin möglich, ohne große Umstände Daten an eine sequentielle Datei anzuhängen. Im anderen Fall (z.B. im 64er-Modus) muß zunächst die gesamte sequentielle Datei in den Speicher des Rechners eingelesen werden, um dann die eingelesenen Daten vor den neuen Daten wieder erneut zu speichern.

Der APPEND-Befehl wird an Stelle des DOPEN-Befehls benutzt, und seine Wirkungsweise zeigen wir am besten mit einem kleinen Beispielprogramm.

```
100 APPEND #2,"TESTDATEI1"
110 IF DS <> 0 THEN PRINTDS$ : END
120 FOR I=101 TO 200
130 PRINT#2,I
140 PRINT I
150 NEXT
160 DCLOSE #2
170 IF DS <> 0 THEN PRINTDS$ : END
```

Listing 4/3.1.4.16-1

Wir haben lediglich die Zeile 100 verändert, wo wir den Begriff DOPEN durch APPEND ersetzt haben. Außerdem wurde das »,W« gestrichen, da es beim APPEND-Befehl klar ist, daß dieser schreibend zugreift. Außerdem wurde in Zeile 120 der Beginn und das Ende unserer Schleife verändert.

Wenn Sie sich nun mit dem dritten oder vierten Beispielprogramm die Datei TESTDATEI anschauen, so hat es den Anschein, als hätten Sie gleich die Zahlen von 1 bis 200 in diese Datei eingetragen.

Sofern Sie die Anzahl der Daten/Datensätze innerhalb der sequentiellen Datei aber im ersten Datum markiert haben, nützt Ihnen der APPEND-Befehl nichts, da Sie dieses Datum nicht ändern können, ohne die gesamte Datei neu zu beschreiben.

4/3.1.4.17

CONCAT — Das Verknüpfen von Dateien

Der APPEND-Befehl ist für das Anhängen von Daten an eine bestehende Datei zuständig, wenn diese per Rechner an die Datei angefügt werden und vom Anwenderprogramm herkommen. Hingegen ist der CONCAT-Befehl zuständig, wenn sich zwei Dateien, die zusammengeschlossen werden sollen, beide auf Diskette befinden. Dieser Befehl kann auch im Direktmodus gegeben werden, und wir wollen ihn dabei gleich für unser Beispiel verwenden.

```
CONCAT "TESTDATEI2" TO "TESTDATEI1"
```

Auch hier können Sie mit unseren Beispielprogrammen zum Einlesen der Daten wieder die Tätigkeit nachvollziehen. Zunächst werden die Zahlen von 1 bis 200 ausgegeben (TESTDATEI1 mit APPEND aus dem letzten Kapitel) und anschließend wiederum die Zahlen von 1 bis 100.

Auch bei Verwendung des CONCAT-Befehls ist es nicht möglich, die Anzahl der Daten/Datensätze zu Beginn der Datei zu notieren, da dieser Merker von der zweiten Datei mitten in die zusammengefaßte Datei übernommen wird.

Beachten Sie beim CONCAT-Befehl, daß Sie zunächst die Datei angeben, die angehängt werden soll, und nicht umgekehrt.

Da CONCAT wie auch APPEND im Programm verwendet werden kann, ergeben sich reichhaltige Möglichkeiten der Behandlung von sequentiellen Dateien, so daß in manchen Fällen auf relative Dateien verzichtet werden kann. Der wichtigste Befehl in bezug auf Direktzugriffsdateien ist RECORD#, den wir abschließend in Kapitel 4/3.1.4 besprechen wollen.

4/3.1.4.18

RECORD — Der Direktzugriff

Der RECORD-Befehl ist derjenige Befehl, der den Direktzugriff erst ermöglicht. Wie bekannt, müssen Sie beim DOPEN-Befehl die Datensatzlänge bei einer Direktzugriffsdatei — mindestens beim ersten Öffnen — angeben. In diesem Fall haben wir es also mit einer sogenannten festen Datensatzlänge (alle Datensätze sind gleich lang) zu tun. Im Gegensatz dazu stehen Datensätze mit variabler Länge die gegenüber denjenigen mit fester Länge, einige Vorteile bieten (Speicherplatzersparnis). Diese lassen sich jedoch mit den zur Verfügung stehenden BASIC-Befehlen nicht realisieren.

Auch eine Direktzugriffsdatei ist nichts anderes als eine sequentielle Datei, jedoch merkt sich der Computer die Datensatzlänge. Wenn Sie nun den 100. Datensatz lesen wollen, müssen 99 Datensätze übersprungen werden. Wenn man dies vereinfacht auf die Anzahl der Zeichen eines Datensatzes reduziert und z.B. eine Datensatzlänge von 250 Byte vorgibt, so beginnt der 100. Datensatz beim 24751. Zeichen.

Bei Angabe der Bytenummer im RECORD-Befehl läßt sich sogar der Datensatzzeiger innerhalb eines Datensatzes positionieren, was wir später auch noch zeigen werden.

Zunächst schreiben wir uns ein kleines Programm, mit dem drei Datensätze in eine relative Datei eingeschrieben werden.

```
100 DOPEN #2,"TESTDATEN",L80
110 IF DS <> 0 THEN PRINTDS$ : END
120 :
130 DATA HUBER,SEPP,WALDSTR. 36,9874 ADORF,123/456789
140 DATA MUELLER,JOSEF,ALLEESTR. 25,1485 BDORF,258/456987
150 DATA SCHMITZ,HANS,RINGSTR. 15,5896 CSTADT,45878/56954
160 :
170 FOR I=1 TO 3
180 READ NA$
190 READ VN$
200 READ SR$
210 READ OT$
220 READ TE$
230 :
240 RECORD #2,I
250 X=DS
260 X$=DS$
```

Listing 4/3.1.4.18-1 (1)

```
270 IF X <> 0 THEN PRINT DS$
280 IF X <> 0 AND X <> 50 THEN END
290 :
300 PRINT#2,NA$;VN$;SR$;OT$;TE$
310 :
320 NEXT
330 :
340 DCLOSE #2
350 IF DS <> 0 THEN PRINT DS$
360 END
60000 SCRATCH"PROGDISK6":PRINTDS$:DSAVE"PROGDISK6":PRINTDS$:END
```

Listing 4/3.1.4.18-1 (2)

Zunächst haben wir in Zeile 100 eine ausreichend lange Datei definiert und stellen in den Zeilen 130 bis 150 einige Datensätze zur Verfügung, die in der FOR . . . NEXT-Schleife ab Zeile 170 eingelesen und der Datei übergeben werden. Dazu wird mittels des RECORD#-Befehls, einer der ersten drei Datensätze angewählt. Zur Sicherheit wird ab Zeile 250 noch eine Fehlerbehandlung durchgeführt, bei der 'kein Fehler/0' und 'RECORD NOT PRESENT/50' erlaubt sind.

Dabei ist es wichtig, daß die Ausgabe der einzelnen Daten (NA$, VN$, SR$, OT$ und TE$) in einer einzigen Zeile untergebracht werden sollten, wobei als Trennung jeweils ein »;« heranzuziehen ist. Achtung: Hinter dem letzten Eintrag kein Semikolon mehr.

Wer Interesse hat, kann auch die Verwendung eines »,« als Trennzeichen austesten bzw. die Verwendung eines »+«, was jedoch der Ausgabe einer einzigen Zeichenreihe gleichkommt.

Daß es sich auch bei Direktzugriffsdateien um sequentielle Dateien handelt, veranschaulicht folgendes Programm, bei dem die Daten fortlaufend — ohne Verwendung des RECORD#-Befehls — von der Datei eingelesen und am Bildschirm ausgegeben werden.

```
100 DOPEN #2,"TESTDATEN",L80
110 IF DS <> 0 THEN PRINTDS$ : END
120 :
130 FOR I=1 TO 3
140 INPUT#2,A$
150 PRINTA$
160 NEXT
170 :
180 DCLOSE#2
```

Listing 4/3.1.4.18-2

Hierbei werden allerdings alle Datensatzelemente ohne irgendwelche Trennzeichen hintereinander ausgegeben.

Mit dem nächsten Programm können Sie die Daten wie bei einer Direktzugriffsdatei einlesen:

```
100 DOPEN #2,"TESTDATEN",L80
110 IF DS <> 0 THEN PRINTDS$ : END
120 :
130 FOR I=1 TO 3
140 RECORD#2,I
150 INPUT#2,NA$,VN$,SR$,OT$,TE$
160 PRINT NA$,VN$,SR$,OT$,TE$
170 NEXT
180 :
190 DCLOSE#2
```

Listing 4/3.1.4.18-3

Und noch ein kurzes Programm, mit dem Sie die Datei zeichenweise auslesen können:

```
100 DOPEN #2,"TESTDATEN",L80
110 IF DS <> 0 THEN PRINTDS$ : END
120 :
130 DO
140 GET#2,A$
150 GETKEY B$
160 IF B$="S" THEN 200
170 PRINTA$,ASC(A$)
180 LOOP
190 :
200 DCLOSE#2
```

Listing 4/3.1.4.18-4

Dies sind jedoch nicht die allgemeingültigen Verwendungsformen von Direktzugriffsdateien, vielmehr wollten wir Ihnen dieses Thema hier etwas näherbringen. Eine optimale Lösung für die Anwendung von Direktzugriffsdateien kann nur anwenderorientiert gegeben werden. Trotzdem wird man in den meisten Fällen bei Direktzugriffsdateien den Weg entsprechend abgelegter Datensatzelemente nehmen oder die Positionierung innerhalb des Datensatzes auf die entsprechende Bytenummer. Letzteres wollen wir Ihnen an einem kleinen Programm verdeutlichen.

```
100 DOPEN #2,"TESTDATEN2",L80
110 IF DS <> 0 THEN PRINTDS$ : END
120 :
130 DATA 1,11,21,41,61
140 :
150 READ NA,VN,SR,OT,TE
160 :
170 DATA HUBER,SEPP,WALDSTR. 36,9874 ADORF,123/456789
180 DATA MUELLER,JOSEF,ALLEESTR. 25,1485 BDORF,258/456987
190 DATA SCHMITZ,HANS,RINGSTR. 15,5896 CSTADT,45878/56954
200 :
210 FOR I=1 TO 3
220 :
230 READ NA$
240 RECORD #2,I,NA
250 PRINT#2,NA$
260 :
270 READ VN$
280 RECORD #2,I,VN
290 PRINT#2,VN$
300 :
310 READ SR$
320 RECORD #2,I,SR
330 PRINT#2,SR$
340 :
350 READ OT$
360 RECORD #2,I,OT
370 PRINT#2,OT$
380 :
390 READ TE$
400 RECORD #2,I,TE
410 PRINT#2,TE$
420 :
430 NEXT
440 :
450 DCLOSE #2
460 IF DS <> 0 THEN PRINT DS$
470 END
```

Listing 4/3.1.4.18-5

Grundlage für dieses Programm war das zuerst in diesem Kapitel vorgestellte Programm. Achten Sie darauf, daß Sie auch in Zeile 100 den Dateinamen geändert haben, um Komplikationen zu vermeiden. Als nächstes haben wir in Zeile 130 fünf Zahlen definiert, die jeweils den Beginn eines Datensatzelementes im Datensatz angeben. Für den Namen sind also zehn Zeichen vorgesehen, ebenfalls für den Vornamen. Straße und Ort können mit jeweils zwanzig Zeichen ausgeführt werden, und für die Telefonnummer steht der Rest (ebenfalls 20 Zeichen) des Datensatzes zur

Verfügung. Diese Daten werden als nächstes eingelesen, worauf die schon bekannten Testdaten folgen.

Innerhalb der FOR . . . NEXT-Schleife ab Zeile 210 werden dann die einzelnen Datensatzelemente eingelesen und entsprechend der Datensatzzeiger positioniert, worauf das Eintragen in den Datensatz erfolgt.

Wenn Sie nun das Programm zum Einlesen einzelner Zeichen von Diskette laufen lassen (Achtung Dateinamen ändern), so werden Sie feststellen, daß nach jedem Eintrag eine Leerzeile am Bildschirm ausgegeben wird, welcher der Code 13 (Carriage Return) zugewiesen ist.

Einlesen können Sie die Daten mit dem gleichen Programmstück, wenn Sie jeweils in den Zeilen 250, 290, 330, 370 und 410 das PRINT durch ein INPUT ersetzen und als Ausgabe auf dem Bildschirm die Zeile 425 einfügen, die wie folgt aussehen könnte:

```
425 PRINT NA$,VN$,SR$,OT$,TE$
```

In unserem Beispiel haben wir jeweils die gesamten Datensätze wieder eingelesen, was meist auch benötigt wird. Wollen Sie nun im zweiten Datensatz (MUELLER) die Telefonnummer ändern, so können Sie einfach schreiben

```
DOPEN#2,"TESTDATEN2"
RECORD#2,2,61
PRINT#2,"12345/6789"
DCLOSE#2
```

Natürlich kann man dies auch noch in ein Programm verpacken, was wir hier jedoch nicht mehr durchführen wollen.

Damit kennen Sie die wichtigsten Grundlagen der Datenbehandlung, und der Programmierung einer eigenen Dateiverwaltung dürfte nichts mehr im Wege stehen.

4/4 Spezielle Programmierthemen

4/4.1.7

Fractale — selbstähnliche Grafiken: Ordnung und Chaos direkt nebeneinander

Im vorliegenden Kapitel wollen wir Grafiken besprechen, die nicht mittels eindeutiger Grafikbefehle gezeichnet, sondern aufgrund einer mathematischen Vorschrift errechnet werden. Gerade bei Fractalen — so heißt der Typ der mathematischen Gebilde und insbesondere deren grafische Auswirkung — ergeben sich interessante mathematische Möglichkeiten. Wir wollen in diesem Kapitel jedoch weitestgehend auf Begriffe wie Attraktoren oder „nicht-ganzzahlig-dimensional" und die mathematischen Hintergründe verzichten und uns schwerpunktmäßig mit der grafischen Auswirkung auseinandersetzen.

Literaturempfehlungen

Wer sich näher mit dieser Thematik befassen möchte, der sei auf ein Buch und zwei Zeitungsartikel hingewiesen:

#1: Pleitgen/Richter: The Beauty Of Fractals, Springer Verlag
#2: Uli Deker/Harry Thomas: Unberechenbares Spiel der Natur — Die Chaos-Theorie, Bild der Wissenschaft, Heft Januar 1983, Seite 62 ff.
#3: Prof. Henning Genz: Fractale, Bild der Wissenschaft, Heft August 1986, Seite 140 (Mathematisches Kabinett)

Besonders das unter #1 genannte Buch sei dem interessierten Leser empfohlen, obwohl es — trotz deutschsprachiger Autoren — vollständig in englisch geschrieben ist.

Ordnung und Chaos

Über 2000 Jahre lang war man der Meinung, daß in der Natur „Ordnung" herrscht. Alle vorkommenden Strukturen wollte man mit wenigen geometrischen Grundformen (z.B. Kreisen oder Kugeln) zusammensetzen können, wenn man diese Grundformen nur beliebig oft und beliebig klein an- und ineinanderschachteln würde. Bei Wolken z.B. hätte man sich hier etwas schwer getan, denn heute weiß man, daß

4/4.1.7.1

Koch-Dreiecke

Bild 4/4.1.7.1-1 stellt ein Bildschirmhardcopy von einer einfachen gezeichneten Linie dar. Da es sich um ein sehr vereinfachtes Beispiel handelt, ist die Verbindung zur Küstenlinie Großbritanniens aus 1000 km Höhe natürlich nicht sehr sinnvoll, aber das Prinzip ist das gleiche.

Rechenvorschrift

Wir wollen nun eine Vorschrift vorgeben, wie diese Linie zu ändern ist. Dazu teilen wir die Linie in drei gleiche Teile und belassen das linke und rechte Drittel der Linie so, wie es ist. Die Länge des mittleren Teilabschnittes wird verdoppelt und als Spitze eines Dreieckes ausgebildet, wie es Bild 4/4.1.7.1-2 zeigt.

Auf die vier Teilabschnitte aus Bild 4/4.1.7.1-2 kann man nun das gleiche Verfahren anwenden und erhält die grafische Darstellung aus Bild 4/4.1.7.1-3.

Jeder der vier Teilabschnitte aus dem letzten Bild ist nicht nur optisch, sondern auch nach Konstruktionsvorschrift identisch mit dem ganzen Linienzug aus Bild 4/4.1.7.1-2, nur etwas kleiner. Wenn man von der Größe und dem Winkel absieht, ist ein Teil aus Bild 4/4.1.7.1-3 nicht von der Linie in Bild 4/4.1.7.1-2 zu unterscheiden. Da sie nicht exakt wegen der Größe und des Winkels übereinstimmen, nennt man sie ähnlich oder selbstähnlich.

Das genannte Konstruktionsprinzip kann man noch häufiger anwenden und es ergeben sich die Linienmuster aus den Bildern 4/4.1.7.1-4 bis 4/4.1.7.1-7, wobei natürlich die Selbstähnlichkeit erhalten bleibt.

Damit wird auch das Beispiel der Küstenlinie Großbritanniens besser deutlich, da bei jedem Durchlauf die Linienlänge um 1/3 gegenüber dem vorherigen Durchlauf verlängert wird.

Auch beim siebten Durchlauf läßt sich die Grundstruktur aus dem zweiten Durchlauf noch gut erkennen, betrachtet man jedoch nur die gezeichnete Linie, so scheint sie etwas chaotisch zu verlaufen, da die Auflösung des Bildschirmes überschritten ist. Hier sieht man auch schon sehr deutlich, daß Ordnung und Chaos sehr nahe beieinander liegen.

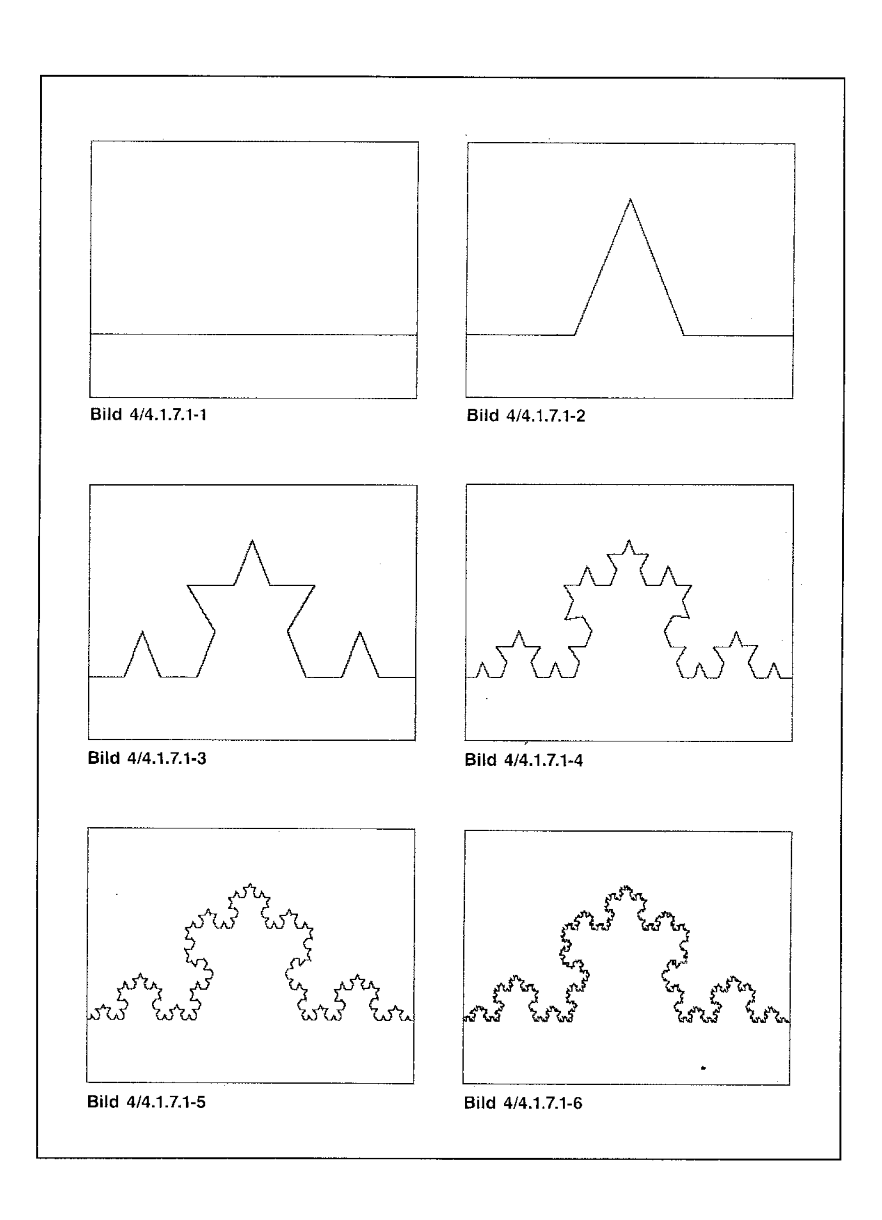


Bild 4/4.1.7.1-1

Bild 4/4.1.7.1-2

Bild 4/4.1.7.1-3

Bild 4/4.1.7.1-4

Bild 4/4.1.7.1-5

Bild 4/4.1.7.1-6

Trotz einfacher Konstruktionsvorschrift sind eine Menge Daten (je Linienstück eine X- und Y-Koordinate) zu verarbeiten. Natürlich wollen wir Ihnen das Listing, mit dem wir diese Koch-Dreiecke gezeichnet haben, nicht vorenthalten (damit das Zeichnen nicht solange dauert, haben wir auf die Basic-Erweiterung aus Kapitel 4/6.4.4 zurückgegriffen):

```
1000 rem ------------------------------
1010 rem ---      koch-dreiecke      ---
1020 rem ------------------------------
1030 :
1040 ma=8000
1050 dim x%(ma)
1060 dim y%(ma)
1070 :
1080 x%(0)=0
1090 y%(0)=150
1100 x%(1)=319
1110 y%(1)=150
1120 :
1130 an=0
1140 fe=1
1150 :
1160 print"Seeee Leertaste -> Weiter
1170 print" RETURN     -> Abbruch
1180 print" H          -> Hardcopy
1190 :
1200 dl=dl+1
1210 if dl=7 then end
1220 :
1230 get a$
1240 if a$=chr$(13) then end
1250 if a$="h" then hrdcpy 0 : goto 1230
1260 if a$<>" " then 1230
1270 :
1280 grafclr
1290 grafon 0,1
1300 :
1310 plot x%(an),y%(an)
1320 :
1330 for i=an+1 to fe
1340 : line x%(i),y%(i)
1350 next
1360 :
1370 n=fe+1
1380 ak=fe
1390 :
1400 for i=an to fe-1
1410 : get a$
1420 : if a$=chr$(13) then end
1430 :
1440 : if x%(i)=x%(i+1) and y%(i)=y%(i+1) then 1650
1450 : ak=ak+1
1460 : x%(ak)=x%(i)
1470 : y%(ak)=y%(i)
1480 :
1490 : ak=ak+1
1500 : x%(ak)=x%(i)+(x%(i+1)-x%(i))/3
1510 : y%(ak)=y%(i)-(y%(i)-y%(i+1))/3
```

Listing 4/4.1.7.1-8 (Teil 1)

```
1520 :
1530 : ak=ak+1
1540 : x%(ak)=x%(ak-1)+(x%(i+1)-x%(i))/6-(y%(i)-y%(i+1))/3
1550 : y%(ak)=y%(ak-1)-(y%(i)-y%(i+1))/6-(x%(i+1)-x%(i))/3
1560 :
1570 : ak=ak+1
1580 : x%(ak)=x%(i)+2*(x%(i+1)-x%(i))/3
1590 : y%(ak)=y%(i)-2*(y%(i)-y%(i+1))/3
1600 :
1610 : ak=ak+1
1620 : x%(ak)=x%(i+1)
1630 : y%(ak)=y%(i+1)
1640 :
1650 next
1660 :
1670 for i=n to ak
1680 : x%(i-n)=x%(i)
1690 : y%(i-n)=y%(i)
1700 next
1710 :
1720 an=0
1730 fe=ak-n
1740 :
1750 goto 1160
1760 :
```

Listing 4/4.1.7.1-8 (Teil 2)

Vorspann der Programme

Zunächst definieren wir uns eine Variable, die uns bei der Felddefinition hilft. Es werden zwei Felder X%( ) und Y%( ) definiert, die später die Anfangs- und Endpunkte der Linien speichern. Vorteilhaft ist es, daß der Endpunkt des einen Linienabschnittes zugleich der Anfangspunkt des nächsten Linienabschnittes ist. Mit den ersten Zuweisungen wird unsere Grundlinie (siehe Bild 4/4.1.7.1-1) in den Feldelementen 0 und 1 festgelegt. Um später Speicherplatz einsparen zu können (die Liniendaten aus den vorherigen Durchgängen werden nicht mehr benötigt) definieren wir uns noch zwei Hilfsvariablen (AN, FE), die uns den jeweils gültigen Index-Bereich in den Feldern X%( ) und Y%( ) markieren.

Menü

Vor den Berechnungen wird noch ein Menü angezeigt, um dem Anwender mitzuteilen, wie er über die Tastatur den Programmablauf beeinflussen kann. Dann wird der Grafikspeicher gelöscht und auf Grafikdarstellung mit der Hintergrundfarbe weiß und der Zeichenfarbe schwarz umgeschaltet.

Ausgabe eines Durchlaufes

Es folgt die Ausgabe des Linienzuges in der Programmschleife von Zeile 1330 bis Zeile 1350.

Berechnung eines Durchlaufes

Während der Benutzer sich das Ergebnis des Durchlaufes ansieht, werden schon die Daten für den nächsten Durchlauf errechnet. Dazu wird eine weitere Hilfsvariable (N für neu) als Zeiger auf den nächsten nicht benutzten Wert in X%( ) und Y%( ) gesetzt und eine zusätzliche Hilfsvariable (AK für aktuell) auf das letzte benutzte Element, da diese Variable sofort in der Schleife erhöht wird.

Ersten Abschnitt berechnen (1450—1470)

In der folgenden Programmschleife erfolgt die Berechnung der Linienabschnitte, nachdem über die Tastatur nach einem Abbruch gefragt wurde. Der Anfangspunkt ist dabei gleich dem Anfangspunkt der Ausgangslinie. Das Ende des ersten Abschnittes liegt ein Drittel der Linienlänge entfernt, wozu zunächst die Linienlänge als Differenz aus Anfang- und Endpunkt errechnet, dann gedrittelt und schließlich zum Anfangswert summiert wird. Dies wird natürlich für beide Koordinaten gleichermaßen durchgeführt.

Zweiten Abschnitt berechnen (1490—1510)

Der Endpunkt des zweiten Abschnittes stellt das rechnerisch größte Problem dar. Am besten veranschaulicht man sich die Vorgehensweise an Bild 4/4.1.7.1-2. Die Spitze liegt in X-Richtung auf genau der Hälfte der gesamten Strecke. Da wir es nicht immer mit einer waagerechten Linie zu tun haben, ziehen wir dazu den Anfangspunkt des zweiten Abschnittes heran und ein Sechstel der Gesamtlänge. Außerdem wird der Y-Anteil des folgenden Streckenzuges der Ausgangslinie mit einem Drittel seines Wertes herangezogen und subtrahiert.

Ähnlich wird die neue Y-Koordinate ausgerechnet, auch hier wird der letzte Summand addiert, da Y=0 am oberen Bildschirmrand zu finden ist.

Dritten und vierten Abschnitt berechnen (1530—1590)

Das Ende des dritten Abschnittes ist wieder recht einfach zu errechnen, da es sich um zwei Drittel der Ausgangslänge handelt und der Endpunkt des ganzen Linienteiles ist gleichzeitig wieder der Anfangspunkt des Nachfolgers der Ausgangslinie.

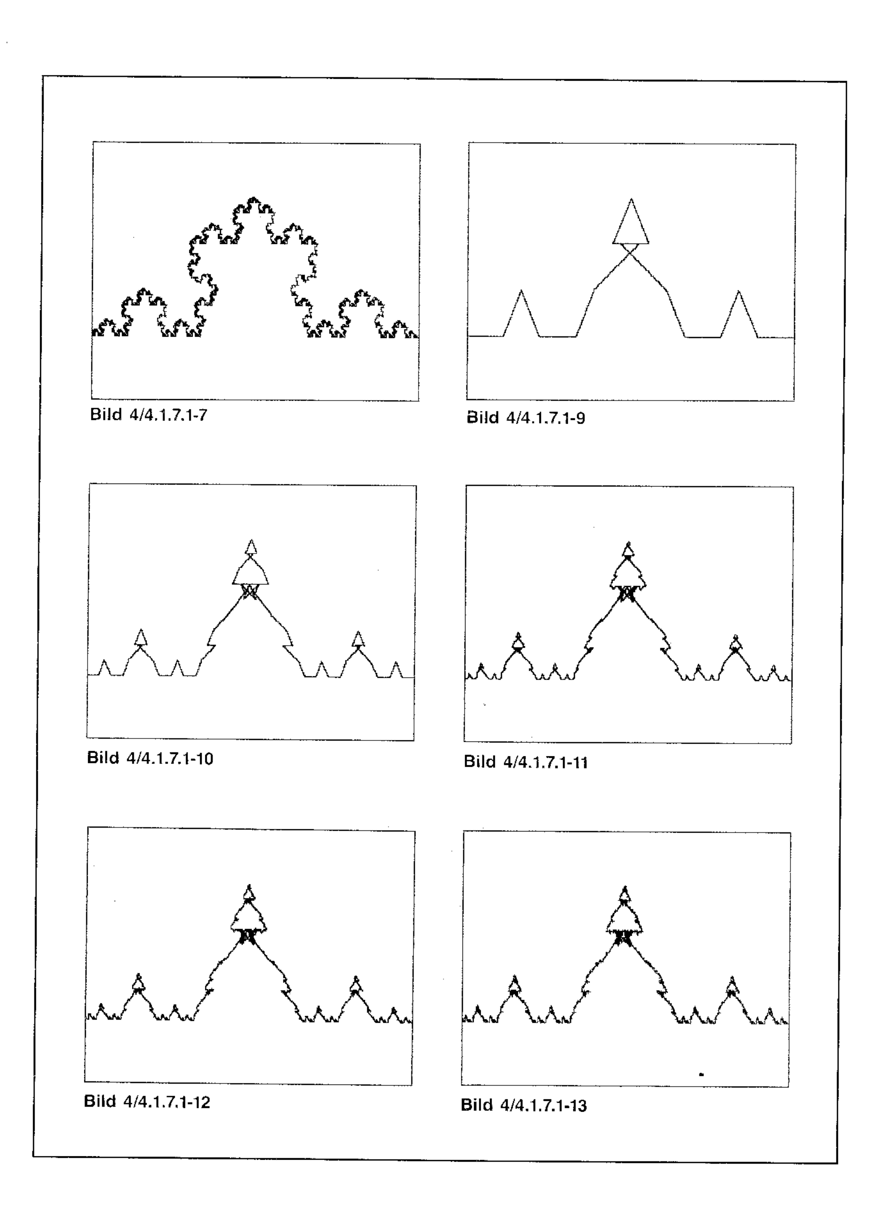


Bild 4/4.1.7.1-7

Bild 4/4.1.7.1-9

Bild 4/4.1.7.1-10

Bild 4/4.1.7.1-11

Bild 4/4.1.7.1-12

Bild 4/4.1.7.1-13

Daten umkopieren

In der nächsten Programmschleife (ab Zeile 1670) werden die neu errechneten Daten in den Feldern X%( ) und Y%( ) an den Anfang geschoben, da die Werte des vorherigen Durchganges nicht mehr benötigt werden. Abschließend sind noch unsere Hilfsvariablen „AN" und „FE" (für Fertig) zu aktualisieren und wir können zum nächsten Lauf übergehen.

Kleine Änderung — große Wirkung

Wenn man das Programm nur geringfügig ändert und bei der Berechnung der X-Position des Endpunktes des zweiten Abschnittes das '+'-Zeichen durch ein '—'-Zeichen ersetzt (Zeile 1540), erhält man die in Bild 4/4.1.7.1-9 bis 4/4.1.7.1-13 dargestellten Linienmuster.

Die ersten beiden Durchläufe wurden nicht abgebildet, da sie mit den Bildern des Ursprungsbeispieles identisch sind. Besonders am siebten Durchlauf kann man erahnen, warum bei computererstellten Zeichentrickfilmen Hintergründe nicht immer explizit gezeichnet werden, sondern fractale Gebilde sind! Trotz der geringen Auflösung unseres Computers erreichen wir mit diesem fractale Muster, die den Umrissen eines Nadelwaldes nicht unähnlich sind.

Noch eine kleine Änderung

Ein weiteres Beispiel dafür, daß man mit Fractalen Landschaftshintergründe darstellen kann, wollen wir ebenfalls mit unseren einfachsten Mitteln — im Gegensatz zu den speziellen Computern für Grafikanimation — darstellen (siehe Bild 4/4.1.7.1-14).

Entgegen unserer Ausgangslinie haben wir hier zwei Linien verwendet, deren Lage unschwer zu erkennen ist. Dieses Muster erreichen wir durch eine ganz kleine Programmänderung:

```
1000 rem -------------------------------
1010 rem ---      koch-dreiecke      ---
1020 rem -------------------------------
1030 :
1040 ma=8000
1050 dim x%(ma)
1060 dim y%(ma)
1070 :
1080 x%(0)= 40
1090 y%(0)=199
1100 x%(1)=170
1110 y%(1)= 10
1111 x%(2)=319
1112 y%(2)=130
1120 :
1130 an=0
1140 fe=2
1150 :
```

Listing 4/4.1.7.1-15

Neben einer zusätzlichen Anfangskoordinate muß noch unsere Hilfsvariable „FE“ umgesetzt werden und wir erhalten die Grafik wieder durch geänderten Parameter bei der Berechnung der dritten X-Koordinate.

Auch mit der Ausgangsform eines Dreieckes (Bild 4/4.1.7.1-16) lassen sich gute Ergebnisse erzielen, wie Bild 4/4.1.7.1-17 und Bild 4/4.1.7.1-18 zeigen. Auch hier haben wir den Anfang des Programmlistings abgebildet.

```
1000 rem -------------------------------
1010 rem ---      koch-dreiecke      ---
1020 rem -------------------------------
1030 :
1040 ma=8000
1050 dim x%(ma)
1060 dim y%(ma)
1070 :
1080 x%(0)=110
1090 y%(0)=150
1100 x%(1)=160
1110 y%(1)= 50
1111 x%(2)=210
1112 y%(2)=150
1113 x%(3)=x%(0)
1114 y%(3)=y%(0)
1120 :
1130 an=0
1140 fe=3
1150 :
```

Listing 4/4.1.7.1-19

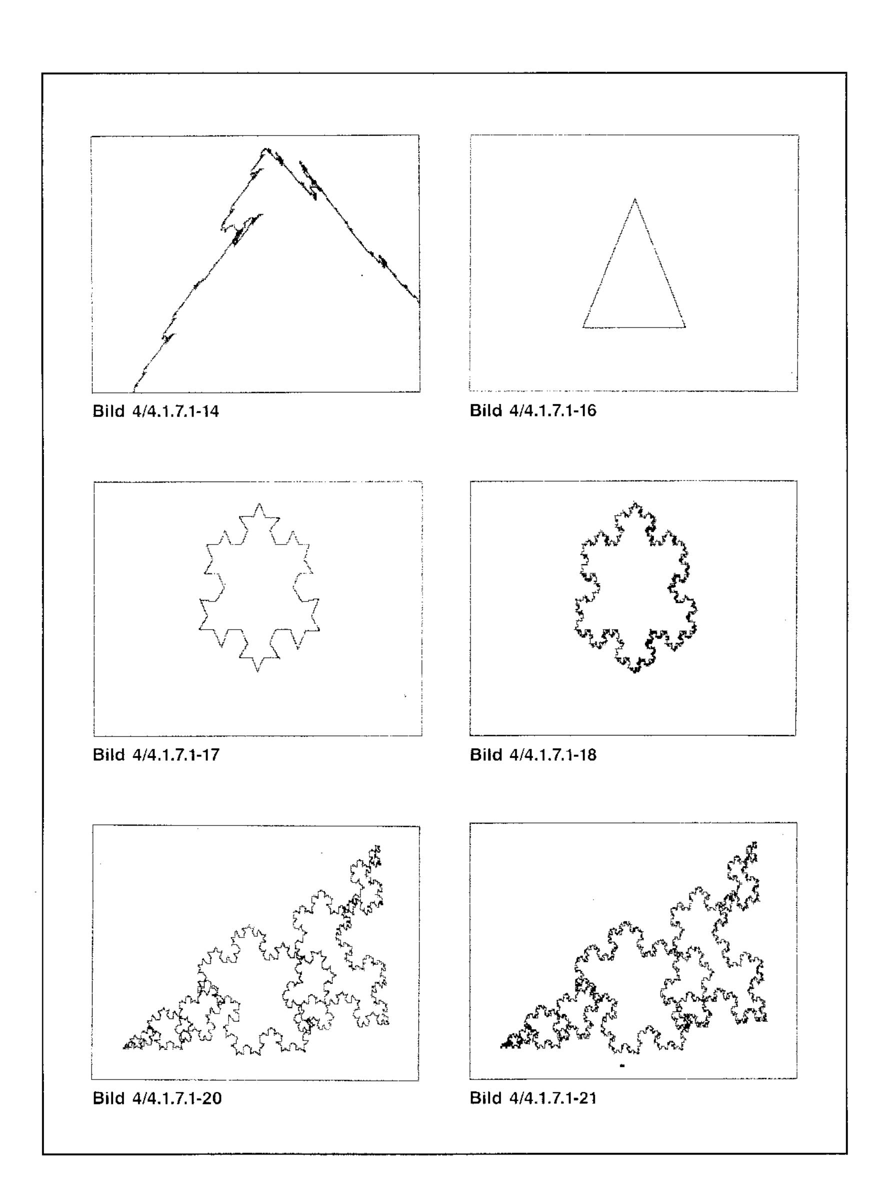


Bild 4/4.1.7.1-14

Bild 4/4.1.7.1-16

Bild 4/4.1.7.1-17

Bild 4/4.1.7.1-18

Bild 4/4.1.7.1-20

Bild 4/4.1.7.1-21

Veränderte Kochdreiecke

Wenn man ein nicht gleichseitiges Dreieck wählt und das Vorzeichen an der bekannten Stelle wieder austauscht, ergeben sich die Ergebnisse aus den Bildern 4/4.1.7.1-20 und 4/4.1.7.1-21, wobei das Ausgangsdreieck unschwer zu erkennen ist.

Wenn Sie nun selbst versuchen, etwas mit den Parametern zu probieren, werden Sie teilweise feststellen, daß sehr kleine Änderungen große Auswirkungen haben, was an einem späteren Beispiel im Rahmen dieses Kapitels noch viel deutlicher wird. Dies ist auch eine besondere Eigenschaft der Fractale: manchmal reichen kleine Änderungen aus, um ein ganz anders geartetes Ergebnis zu erhalten.

Schmetterlingseffekt

Edward N. Lorenz — ein Meteorologe — nannte dies 1963 den Schmetterlingseffekt: der Flügelschlag eines Schmetterlings sei in der Lage, das Wettergeschehen langfristig zu verändern.

4/4.1.7.2

Sierpinski-Dreieck

In etwa ähnlich den Koch-Dreiecken ist das Sierpinski-Dreieck. Ausgangsform ist ein gleichseitiges Dreieck, wie es in Bild 4/4.1.7.2-1 dargestellt ist.

Änderungsvorschrift

Durch die Mittelpunkte der Seitenlinien wird ein neues Dreieck beschrieben, das in das Ausgangsdreieck eingelagert werden kann, wie Durchgang 1 (Bild 4/4.1.7.2-2) zeigt. Dabei wird das Ausgangsdreieck in vier gleichseitige Dreiecke zerlegt. Die Dreiecke an den Spitzen des Ausgangsdreiecks werden nach dem gleichen Verfahren behandelt, wie die anderen Durchgänge in Bild 4/4.1.7.2-3 bis Bild 4/4.1.7.2-7 zeigen. Auch das Sierpinski-Dreieck ist selbstähnlich.

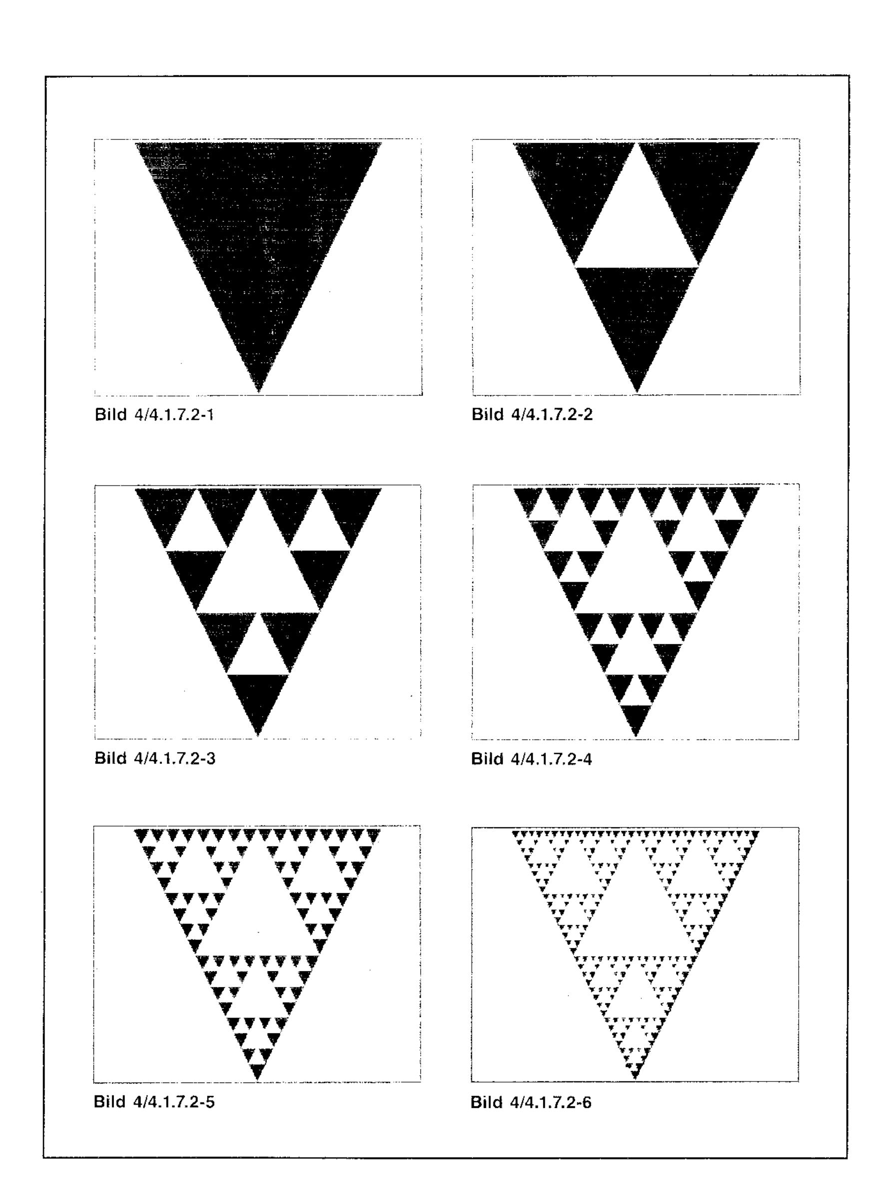


Bild 4/4.1.7.2-1

Bild 4/4.1.7.2-2

Bild 4/4.1.7.2-3

Bild 4/4.1.7.2-4

Bild 4/4.1.7.2-5

Bild 4/4.1.7.2-6

Natürlich wollen wir Ihnen das Programmlisting nicht vorenthalten, das die verschiedenen Durchgänge am Sierpinski-Dreieck errechnet:

```
1000 rem ------------------------------
1010 rem ---   sierpinski-dreieck   ---
1020 rem ------------------------------
1030 :
1040 ma=6000
1050 dim x%(ma)
1060 dim y%(ma)
1070 dim xa%(ma/3)
1080 dim ya%(ma/3)
1090 :
1100 z=0
1110 :
1120 x%(0)= 40
1130 y%(0)=  3
1140 dx   =120
1150 dy   =dx*0.81
1160 :
1170 grafclr
1180 grafon 0,1
1190 drawmode 0
1200 :
1210 for t=x%(0) to x%(0)+2*dx
1220 : plot t,y%(0)
1230 : line x%(0)+dx,y%(0)+2*dy
1240 next:drawmode 1
1250 :
1260 for i=0 to z
1270 : xa%(i)=x%(i)
1280 : ya%(i)=y%(i)
1290 next
1300 :
1310 for i=0 to z
1320 : for t=xa%(i)+dx/2 to xa%(i)+3*dx/2
1330 :   plot xa%(i)+dx,ya%(i)
1340 :   line t,ya%(i)+dy
1350 : next
1360 :
1370 : x%(3*i)=xa%(i)
1380 : y%(3*i)=ya%(i)
1390 : x%(3*i+1)=xa%(i)+dx
1400 : y%(3*i+1)=ya%(i)
1410 : x%(3*i+2)=xa%(i)+dx/2
1420 : y%(3*i+2)=ya%(i)+dy
1430 next
1440 :
1450 z=3*z+2
1460 dx=dx/2
1470 dy=dy/2
1480 :
1490 get a$
1500 if a$=chr$(13) then end
1510 if a$=" " then1260
1520 goto1490
1530 :
```

Listing 4/4.1.7.2-8

Vorspann

Anders als bei den Koch-Dreiecken wählen wir hier vier Felder, jeweils für die neu zu errechnenden Werte (X%( ), Y%( )) und die Werte des vorherigen Durchganges (XA%( ), YA%( ); A für alt). Außerdem benötigen wir einen Zähler und die Position der linken oberen Ecke des Dreieckes, die anderen Ecken errechnen wir aus der Differenz in X- und Y-Richtung (DX und DY). Die Differenz benötigen wir später noch für unsere Berechnungen, so daß wir sie gleich zu Anfang verwenden können. Der Faktor bei der Differenz in Y-Richtung korrigiert diesmal die unterschiedlichen Pixelabstände der beiden Richtungen auf dem Bildschirm, so daß wir nicht rechnerisch, sondern optisch gleichseitige Dreiecke erhalten. Nach Festlegen der Rahmenbedingungen und Einschalten der Grafik zeichnen wir das Ausgangsdreieck, wobei die Schleife in den Zeilen 1210 bis 1240 zum Zeichnen eines ausgefüllten Dreieckes dient.

Berechnungen

Zu Beginn aller Berechnungen werden die Daten des zuletzt durchgerechneten Durchlaufes in die Felder XA%( ) und YA%( ) übertragen (Zeilen 1180 bis 1210). In der Programmschleife ab 1230 bis Zeile 1350 wird jedes Dreieck gezeichnet. Anschließend werden die linken oberen Eckpunkte der drei neuen Dreiecke errechnet und in X%( ) und Y%( ) abgelegt. Nach einem Durchlauf wird der Zeiger auf das letzte aktuelle Element gesetzt und die Differenzwerte halbiert.

4/4.1.7.3

Fractale Bäume

Wenn man sich mit Fractalen beschäftigt, möchte man natürlich auch selbst etwas herumprobieren. Ist Ihnen bewußt, daß Pflanzen auch selbstähnlich sind? Blumenliebhaber nutzen diesen Umstand, indem sie bei einigen Pflanzen Seitentriebe abschneiden, einpflanzen und so eine neue Pflanze erhalten, die der alten ähnlich ist.

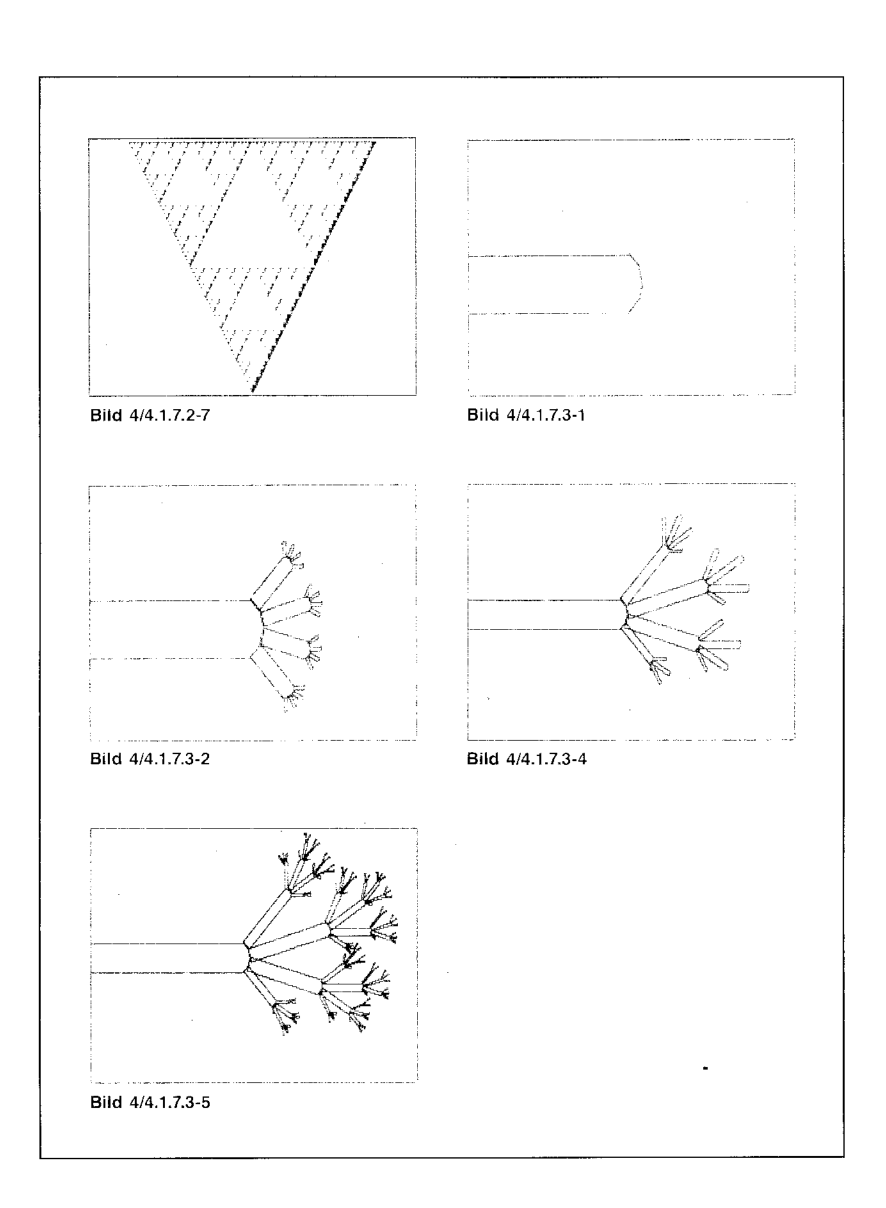


Bild 4/4.1.7.2-7

Bild 4/4.1.7.3-1

Bild 4/4.1.7.3-2

Bild 4/4.1.7.3-4

Bild 4/4.1.7.3-5

Ein Beispiel

Wenn man — stark verallgemeinert — aus einem Baumstamm vier Äste herauswachsen läßt und aus diesen Ästen jeweils wieder vier Zweige, so ergibt sich der in Bild 4/4.1.7.3-1 dargestellte „Baum". Auch dies ist wieder ein Beispiel dafür, daß sich die Natur sehr gut durch Fractale darstellen läßt, wenn man als Heimcomputerbesitzer auch immer an der Kapazität des Rechners scheitert und nur einen schematischen Überblick erhält.

Natürlich geht bei der Erstellung von fractalen Grafiken die meiste Zeit für das Ausprobieren der richtigen Parameter drauf und die wenigste Zeit für das Programmieren.

Im folgenden Listing zu den fractalen Bäumen möchten wir Ihnen eine andere Möglichkeit vorstellen, wie Sie speicherplatzschonend auch kompliziertere Grundformen bearbeiten können.

Der geneigte Leser möge — auf Kosten der Zahl der Durchgänge — versuchen, auch ein paar seitliche Äste an den Baum aus dem letztgenannten Bild anzufügen. Hier zunächst das Listing:

```
1000 rem ------------------------------
1010 rem ---     fractale baeume    ---
1020 rem ------------------------------
1030 :
1040 ma=3000
1050 dim x%(ma)
1060 dim y%(ma)
1070 dim b%(ma)
1080 dim d(ma)
1090 :
1100 x%(0)= 0
1110 y%(0)=90
1120 b%(0)=45
1130 d(0) =0
1140 :
1150 input "Tiefe (1-5)";t
1160 if t<1 or t>5 then 1150
1170 :
1180 for i=0 to t-1
1190 : su=su+4↑i
1200 next
1210 :
1220 grafclr
1230 grafon 0,1
1240 :
1250 for i=0 to summe-1
1260 : get a$
1270 : if a$=chr$(13) then end
1280 :
```

Listing 4/4.1.7.3-3 (Teil 1)

```
1290 : gosub 2000
1300 next
1310 :
1320 get a$
1330 if a$="" then 1320
1340 :
1350 end
1360 :
2000 rem -------------------------------
2010 rem --- upro: baum zeichnen     ---
2020 rem -------------------------------
2030 :
2040 x=x%(i)
2050 y=y%(i)
2060 b=b%(i)
2070 d=d(i)
2080 :
2090 xa=x
2100 ya=y
2110 :
2120 plot x,y
2130 xn=x+b*sin(d)
2140 yn=y+b*cos(d)
2150 line xn,yn
2160 :
2170 xn=xn+3.5*b*sin(d+π/2)
2180 yn=yn+3.5*b*cos(d+π/2)
2190 line xn,yn
2200 :
2210 xn=xn+0.3*b*sin(d+3*π/4)
2220 yn=yn+0.3*b*cos(d+3*π/4)
2230 line xn,yn
2240 :
2250 a=a+1
2260 if a>su then 2330
2270 :
2280 x%(a)=xn
2290 y%(a)=yn
2300 b%(a)=0.3*b
2310 d(a)=d-π/4
2320 :
2330 xn=xn+0.3*b*sin(d+11*π/12)
2340 yn=yn+0.3*b*cos(d+11*π/12)
2350 line xn,yn
2360 :
2370 a=a+1
2380 if a>su then 2450
2390 :
2400 x%(a)=xn
2410 y%(a)=yn
2420 b%(a)=0.3*b
2430 d(a)=d-1*π/12
2440 :
2450 xn=xn+0.3*b*sin(d+13*π/12)
2460 yn=yn+0.3*b*cos(d+13*π/12)
2470 line xn,yn
2480 :
2490 a=a+1
2500 if a>su then 2570
2510 :
2520 x%(a)=xn
2530 y%(a)=yn
2540 b%(a)=0.3*b
2550 d(a)=d+π/12
```

Listing 4/4.1.7.3-3 (Teil 2)

```
2560 :
2570 xn=xn+0.3*b*sin(d+5*π/4)
2580 yn=yn+0.3*b*cos(d+5*π/4)
2590 line xn,yn
2600 :
2610 a=a+1
2620 if a>su then 2690
2630 :
2640 x%(a)=xn
2650 y%(a)=yn
2660 b%(a)=0.3*b
2670 d(a)=d+π/4
2680 :
2690 line xa,ya
2700 :
2710 return
2720 :
```

Listing 4/4.1.7.3-3 (Teil 3)

Ähnlich wie beim Sierpinski-Dreieck wollen wir unsere Baum-Grundstruktur aufgrund eines einzigen Punktes, der Breite und des Drehwinkels berechnen. Die Winkel müssen beim C 64 im Bogenmaß angegeben werden.

Zu Beginn geben wir wieder ein maximales Feldelement vor. X%( ) und Y%( ) geben die Position des Basispunktes an, B%( ) die Breite und D( ) den Drehwinkel. Dann wird der erste Punkt festgelegt und anschließend die Anzahl der Durchgänge (Tiefe) erfragt. In der Schleife ab Zeile 1180 rechnen wir aus, welches Element der vorgenannten Felder wir als letztes benutzen, um dann so oft die Prozedur zum Zeichnen aufzurufen.

Prozedur: Baumteil zeichnen

Der besseren Übersicht wegen weisen wir den Variablen X, Y, B und D die aktuellen Werte des Teilstückes zu, zu dem die vier neuen Baumteile errechnet werden sollen. Wie Sie aus Bild 4/4.1.7.3-1 sehen, sind am oberen Teil der Struktur vier abgeschrägte Stücke, die beiden äußeren im 45°-Winkel und die beiden inneren im 15°-Winkel. Auf jeden dieser Teile wollen wir eine neue, identische Struktur aufsetzen. Die Ausgangsdaten dazu entnehmen wir jeweils dem aktuellen Feldelement.

Zu Beginn merken wir uns die Ausgangskoordinaten, da die letzte Linie wieder dorthin gezogen wird. Anschließend wird der Endpunkt der ersten Linie berechnet und eine Linie zu diesem Punkt ausgegeben. Wie man an den Koordinaten ersehen kann, zeichnen wir ihn quer, d.h. die Wurzel befindet sich an der linken Bildschirmseite.

Dadurch, daß wir die Winkelfunktionen Sinus und Cosinus zu Hilfe nehmen, können wir uns die Speicherung der einzelnen Eckpunkte der Grundstruktur ersparen und uns auch innerhalb der Auswahl der Drehwinkel freier bewegen. Achten Sie besonders auf die verwendeten Parameter, z.B. den Wert „6,5" als Multiplikator für die Breite. Diese Werte können Sie für eigenes Probieren später ändern. Der genannte Wert sagt nichts anderes aus, als daß die Höhe der Struktur dem 6,5-fachen der Breite entspricht. Auch die Schrägen basieren in ihrer Breite auf der Ausgangsbasis, nämlich einem Drittel.

Die Hilfsvariable A dient zum Weiterzählen der aktuell zu besetzenden Feldelemente und außerdem als Abbruchkriterium. Bei unseren vier schrägen Abschnitten ist der Endpunkt der Linie jeweils die Basis für ein neues Element. Der Drehwinkel der neuen Figur ergibt sich natürlich auch aufgrund des Drehwinkels der vorhandenen Figur und dem Winkel des neuen Abschnittes (2*Pi ist ein Vollkreis = 360°). Die — etwas umständliche — Vorgehensweise mit Berechnung der neuen Koordinaten in Variablen und nicht sofort beim Ziehen der Linien, brauchen wir, um die Daten für die Äste zu ermitteln.

Und wieder wollen wir ein kleines Beispiel dafür liefern, daß kleine Änderungen eine ganz andere optische Auswirkung haben. Behalten Sie nur beim rechten der vier Äste den Faktor 0.3 bei und ändern sie die anderen Breiten nacheinander auf 0.5, 0.55 und 0.4. Dies gilt natürlich nur für die Faktoren bei Übergabe für neue Astdaten und nicht beim Zeichnen. Das Ergebnis ist in den Bildern 4/4.1.7.3-4 und 4/4.1.7.3-5 dargestellt und hat natürlich schon etwas mehr Ähnlichkeit mit einem Baum als unser Ausgangsbeispiel.

Das Programmm befindet sich auf der Grundwerksdiskette

4/4.2

Sound beim C 64

Neben den Sprites und dem VIC bildet der SID (Sound-Interface-Device) eine weitere hervorragende Eigenschaft des Commodore 64. Bei den bisherigen Commodore-Geräten der Serien 2000/3000/4000/8000 konnte — wenn überhaupt — ein einfacher „Piepser" über ein Schieberegister gesteuert werden. Computer-Enthusiasten haben mit wenigen Registern auch hier wunderbare Geräuscheffekte erzielt bzw. sogar Musik erklingen lassen.

Der Commodore 64 bringt mit seinen SID drei Stimmen (Ton-Generatoren) mit einer Menge Register, so daß man in dieser Hinsicht den Computer auch als Synthesizer bezeichnen kann.

Im folgenden wollen wir auf die Eigenschaften des SID eingehen, seine Möglichkeiten und die Art der Programmierung erläutern. Eine Aufstellung seiner Register finden Sie in Kapitel 3/1.2.4.

Dazu wird zunächst ein Programm (SOUNDTEST) vorgestellt, mit dem Sie die Register beliebig einstellen können und dazu der entsprechende Ton erklingt. Dieses Programm können Sie sehr gut dazu nutzen, eigene Möglichkeiten zu entdecken bzw. gewünschte Klangmöglichkeiten auszutesten.

4/4.2.1

Soundtest

Das Programm SOUNDTEST soll Ihnen dabei helfen, die Klangmöglichkeiten Ihres Commodore 64 kennenzulernen. Sie können alle Register des SID einzeln beeinflussen, wobei Sie auch jeweils den Ton ein- oder ausschalten können. Zum besseren Verständnis der Begriffe wie ATTACK, DECAY, SUSTAIN und RELEASE sei auf die Grafik in Bild 4/4.2.1-1 verwiesen.

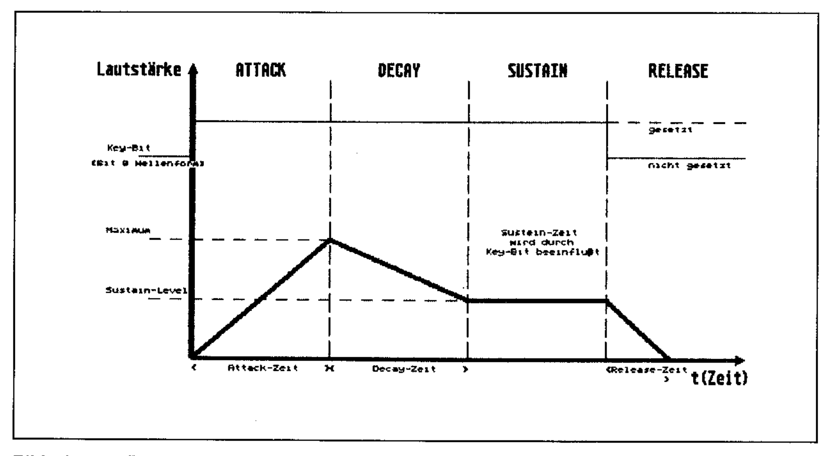


Bild 4/4.2.1-1 Darstellung der Begriffe ATTACK, DECAY, SUSTAIN und RELEASE

```
1000 REM ----------------------------------------------------------------
1005 REM ---                    S O U N D - T E S T                   ---
1010 REM ----------------------------------------------------------------
1015 :
1020 REM ----------------------------------------------------------------
1025 REM ---          VORSPANN UND BESETZEN DER VARIABLEN             ---
1030 REM ----------------------------------------------------------------
1035 :
1040 BL$ = "                                        "
```

Listing 4/4.2.1-1 (Teil 1)

```
1045 CL$ = "[illegible]"
1050 R$  = "[illegible]"
1055 D$  = "[illegible]"
1060 :
1065 DIM S$(8)
1070 DIM FL(3),FH(3),PL(3),PH(3),WE(3),AD(3),SR(3)
1075 DIM ST$(44),P$(44),W(24),NS(10),NA(10)
1080 :
1085 POKE 53280,14
1090 POKE 53281,7
1095 :
1100 REM ---              BESCHREIBUNG DER WELLENFORMEN              ---
1105 :
1110 WL$(8)="RAU  "
1115 WL$(7)="RECH "
1120 WL$(6)="SAEG "
1125 WL$(5)="DREI "
1130 WL$(4)="TEST "
1135 WL$(3)="RING "
1140 WL$(2)="SYNC "
1145 WL$(1)="KEY  "
1150 :
1155 REM ---              REGISTERVARIABLEN BESETZEN                 ---
1160 :
1165 SI   =54272
1170 FL(1)=SI
1175 FL(2)=SI+7
1180 FL(3)=SI+14
1185 FH(1)=SI+1
1190 FH(2)=SI+8
1195 FH(3)=SI+15
1200 PL(1)=SI+2
1205 PL(2)=SI+9
1210 PL(3)=SI+16
1215 PH(1)=SI+3
1220 PH(2)=SI+10
1225 PH(3)=SI+17
1230 WE(1)=SI+4
1235 WE(2)=SI+11
1240 WE(3)=SI+18
1245 AD(1)=SI+5
1250 AD(2)=SI+12
1255 AD(3)=SI+19
1260 SR(1)=SI+6
1265 SR(2)=SI+13
1270 SR(3)=SI+20
1275 IL=SI+21
1280 IH=SI+22
1285 RF=SI+23
1290 LA=SI+24
1295 S=1
1300 :
1305 ST$(1)="TON AUS"
1310 ST$(2)="*"
1315 :
1320 REM ---                  CURSORPOSITIONIERUNG                   ---
1325 :
1330 P$(1)="[illegible]"+LEFT$(R$,26)+"          [illegible]"
1335 P$(2)="[illegible]"+LEFT$(D$,2)+LEFT$(R$,17)+" [illegible]"
1340 P$(3)="[illegible]"+LEFT$(D$,2)+LEFT$(R$,27)+" [illegible]"
1345 P$(4)="[illegible]"+LEFT$(D$,2)+LEFT$(R$,37)+" [illegible]
1350 P$(5)="[illegible]"+LEFT$(D$,3)+LEFT$(R$,10)+"      [illegible]"
1355 P$(6)="[illegible]"+LEFT$(D$,3)+LEFT$(R$,20)+"      [illegible]"
1360 P$(7)="[illegible]"+LEFT$(D$,3)+LEFT$(R$,30)+"      [illegible]"
```

Listing 4/4.2.1-1 (Teil 2)

```
1365 P$(8)="█"+LEFT$(D$,4)+LEFT$(R$,10)+"      █"
1370 P$(9)="█"+LEFT$(D$,4)+LEFT$(R$,20)+"      █"
1375 P$(10)="█"+LEFT$(D$,4)+LEFT$(R$,30)+"      █"
1380 P$(11)="█"+LEFT$(D$,5)+LEFT$(R$,15)+"     █"
1385 P$(12)="█"+LEFT$(D$,5)+LEFT$(R$,25)+"     █"
1390 P$(13)="█"+LEFT$(D$,5)+LEFT$(R$,35)+"     █"
1395 P$(14)="█"+LEFT$(D$,6)+LEFT$(R$,15)+"     █"
1400 P$(15)="█"+LEFT$(D$,6)+LEFT$(R$,25)+"     █"
1405 P$(16)="█"+LEFT$(D$,6)+LEFT$(R$,35)+"     █"
1410 P$(17)="█"+LEFT$(D$,7)+LEFT$(R$,15)+"     █"
1415 P$(18)="█"+LEFT$(D$,7)+LEFT$(R$,25)+"     █"
1420 P$(19)="█"+LEFT$(D$,7)+LEFT$(R$,35)+"     █"
1425 P$(20)="█"+LEFT$(D$,8)+LEFT$(R$,15)+"     █"
1430 P$(21)="█"+LEFT$(D$,8)+LEFT$(R$,25)+"     █"
1435 P$(22)="█"+LEFT$(D$,8)+LEFT$(R$,35)+"     █"
1440 P$(29)="█"+LEFT$(D$,11)+LEFT$(R$,35)+"     █"
1445 P$(30)="█"+LEFT$(D$,12)+LEFT$(R$,0)
1450 P$(30)=P$(30)+LEFT$(BL$,30)+LEFT$(CL$,30)
1455 P$(31)="█"+LEFT$(D$,13)+LEFT$(R$,35)+"     █"
1460 P$(32)="█"+LEFT$(D$,14)+LEFT$(R$,0)
1465 P$(32)=P$(32)+LEFT$(BL$,30)+LEFT$(CL$,30)
1470 P$(33)="█"+LEFT$(D$,15)+LEFT$(R$,35)+"     █"
1475 P$(34)="█"+LEFT$(D$,16)+LEFT$(R$,0)
1480 P$(34)=P$(34)+LEFT$(BL$,30)+LEFT$(CL$,30)
1485 P$(23)="█"+LEFT$(D$,9)+LEFT$(R$,15)+"     █"
1490 P$(24)="█"+LEFT$(D$,9)+LEFT$(R$,25)+"     █"
1495 P$(25)="█"+LEFT$(D$,9)+LEFT$(R$,35)+"     █"
1500 P$(26)="█"+LEFT$(D$,10)+LEFT$(R$,15)+"     █"
1505 P$(27)="█"+LEFT$(D$,10)+LEFT$(R$,25)+"     █"
1510 P$(28)="█"+LEFT$(D$,10)+LEFT$(R$,35)+"     █"
1515 P$(42)="█"+LEFT$(D$,21)+LEFT$(R$,35)+"     █"
1520 P$(44)="█"+LEFT$(D$,22)+LEFT$(R$,20)+"     █"
1525 P$(35)="█"+LEFT$(D$,17)+LEFT$(R$,15)+"       █"
1530 P$(36)="█"+LEFT$(D$,17)+LEFT$(R$,30)+"       █"
1535 P$(37)="█"+LEFT$(D$,18)+LEFT$(R$,14)+"     █"
1540 P$(38)="█"+LEFT$(D$,18)+LEFT$(R$,35)+"     █"
1545 P$(39)="█"+LEFT$(D$,19)+LEFT$(R$,35)+"     █"
1550 P$(40)="█"+LEFT$(D$,20)+LEFT$(R$,4)+"     █"
1555 P$(41)="█"+LEFT$(D$,20)+LEFT$(R$,15)+"     █"
1560 P$(43)="█"+LEFT$(D$,22)+LEFT$(BL$,13)+LEFT$(CL$,13)
1565 :
2000 REM -------------------------------------------------------------
2010 REM ---                   HAUPTPROGRAMM                       ---
2020 REM -------------------------------------------------------------
2030 :
2040 REM ---          INFORMATIONSTABLEAU ANZEIGEN                  ---
2050 :
2060 GOSUB 3100
2070 GET A$
2080 :
2090 REM ---                  SPRUNGVERTEILER                       ---
2100 :
2110 IF A$="I" THEN GOSUB 3100
2120 IF A$="T" THEN GOSUB 3640
2130 IF A$="P" THEN GOSUB 4040
2140 IF A$="W" THEN GOSUB 4340
2150 IF A$="A" THEN GOSUB 4740
2160 IF A$="D" THEN GOSUB 4940
2170 IF A$="S" THEN GOSUB 5140
2180 IF A$="R" THEN GOSUB 5340
2190 IF A$="F" THEN GOSUB 5840
2200 IF A$="L" THEN GOSUB 5540
2210 IF A$="1" OR A$="2" OR A$="3" THEN GOSUB 7240
2220 IF A$=" " THEN GOSUB 7440
```

Listing 4/4.2.1-1 (Teil 3)

```
2230 :
2240 GOTO 2070
2250 :
3000 REM ----------------------------------------------------------------
3010 REM ----------------------------------------------------------------
3020 REM ---                    UNTERPROGRAMME                        ---
3030 REM ----------------------------------------------------------------
3040 REM ----------------------------------------------------------------
3050 :
3060 REM ----------------------------------------------------------------
3070 REM ---                  INFORMATIONSTABLEAU                     ---
3080 REM ----------------------------------------------------------------
3090 :
3100 PRINT"          INFORMATION"
3110 PRINT
3120 PRINT "SIE HABEN FOLGENDE MOEGLICHKEITEN:"
3130 PRINT "I=DAMIT KOENNEN SIE ZWISCHEN DIESEM
3140 PRINT "  HILFSTABLEAU UND DER REGISTERANZEIGE"
3150 PRINT "  HIN UND HER SCHALTEN
3160 PRINT "T=TONHOEHE"
3170 PRINT "P=PULSVERHAELTNIS"
3180 PRINT "A=ATTACK-ZEIT"
3190 PRINT "D=DECAY-ZEIT"
3200 PRINT "S=SUSTAIN-LEVEL"
3210 PRINT "R=RELEASE-ZEIT"
3220 PRINT "W=WELLENFORM"
3230 PRINT "1=1.STIMME,DIE SIE EINSTELLEN"
3240 PRINT "2=2.STIMME,DIE SIE EINSTELLEN"
3250 PRINT "3=3.STIMME,DIE SIE EINSTELLEN"
3260 PRINT "F=FILTER,DABEI HABEN SIE FOLGENDE WAHL:"
3270 PRINT "  F=FILTERECKFREQUENZ"
3280 PRINT "  R=RESONANZ"
3290 PRINT "  A=FILTERART"
3300 PRINT "  Q=QUELLE,DIE UEBER DEN FILTER LAEUFT"
3310 :
3320 GET A$
3330 IF A$="" OR A$<>"I" THEN GOTO 3320
3340 :
3350 PRINT "";
3360 PRINT "     S I D  TESTPROGRAMM  "
3370 PRINT TAB(10);"STIMME1";TAB(20);"STIMME2";TAB(30);"STIMME3"
3380 PRINT "TONHOEHE :"
3390 PRINT "PARAMETER:"
3400 PRINT "LOW BYTE :(0)=";TAB(20);"(7)=";TAB(30);"(14)="
3410 PRINT "HIGH BYTE:(1)=";TAB(20);"(8)=";TAB(30);"(14)="
3420 PRINT "PULS LOW :(2)=";TAB(20);"(9)=";TAB(30);"(15)="
3430 PRINT "PULS HIGH:(3)=";TAB(20);"(10)=";TAB(30);"(16)="
3440 PRINT "ATT./DEC. :(5)=";TAB(20);"(12)=";TAB(30);"(18)="
3450 PRINT "SUS./REL. :(6)=";TAB(20);"(13)=";TAB(30);"(19)="
3460 PRINT TAB(5);"WELLENFORM STIMME 1:";TAB(30);"(4)="
3470 PRINT TAB(5);"WELLENFORM STIMME 2:";TAB(30);"(11)="
3480 PRINT TAB(5);"WELLENFORM STIMME 3:";TAB(30);"(18)="
3490 PRINT "FILTERFREQUENZ=      HZ";TAB(24);"PAR.:"
3500 PRINT "LOW BYTE:(21)=";TAB(19);"HIGH BYTE:";TAB(30);"(22)="
3510 PRINT TAB(2);"RESONANZ";TAB(16);"QUELLE";TAB(30);"(23)="
3520 PRINT TAB(2);"FILTERART";TAB(16);"LAUTSTAERKE";TAB(30);"(24)="
3530 :
3540 FOR I=1 TO 44
3550 : PRINT P$(I);ST$(I)
3560 NEXT
3570 :
3580 RETURN
3590 :
```

Listing 4/4.2.1-1 (Teil 4)

```
3600 REM -------------------------------------------------------------------
3610 REM ---                    TONHOEHE EINSTELLEN                      ---
3620 REM -------------------------------------------------------------------
3630 :
3640 PRINT "██"+LEFT$(D$,23)+LEFT$(BL$,39)+"□"
3650 INPUT "TONHOEHE IN HZ(0-3846)";A
3660 IF A<0 OR A>3846 THEN GOTO 3640
3670 :
3680 NS(1)=4+S
3690 ST$(4+S)=STR$(A)
3700 P=INT(A/0.0587)
3710 :
3720 NS(2)=7+S
3730 ST$(7+S)=STR$(P)
3740 H=INT(P/255)
3750 :
3760 NS(3)=13+S
3770 ST$(13+S)=STR$(H)
3780 L=INT(P-H*255)
3790 :
3800 NS(4)=10+S
3810 ST$(10+S)=STR$(L)
3820 ZS=4
3830 :
3840 NA(1)=S*7-7
3850 W(S*7-7)=L
3860 NA(2)=S*7-6
3870 W(S*7-6)=H
3880 ZA=2
3890 :
3900 GOSUB 7740
3910 :
3920 RETURN
3930 :
4000 REM -------------------------------------------------------------------
4010 REM ---                    PULS EINSTELLEN                          ---
4020 REM -------------------------------------------------------------------
4030 :
4040 PRINT "██"+LEFT$(D$,23)+LEFT$(BL$,39)+"□"
4050 INPUT "PULSPARAMETER(0-4095)";A
4060 IF A<0 OR A>4095 THEN GOTO 4040
4070 :
4080 H=INT(A/256)
4090 L=INT(A-H*256)
4100 :
4110 NS(1)=16+S
4120 ST$(16+S)=STR$(L)
4130 :
4140 NS(2)=19+S
4150 ST$(19+S)=STR$(H)
4160 ZS=2
4170 :
4180 NA(1)=S*7-5
4190 W(S*7-5)=L
4200 :
4210 NA(2)=S*7-4
4220 W(S*7-4)=H
4230 ZA=2
4240 :
4250 GOSUB 7740
4260 :
4270 RETURN
4280 :
```

Listing 4/4.2.1-1 (Teil 5)

```
4300 REM -----------------------------------------------------------------
4310 REM ---                     WELLENFORM EINSTELLEN                     ---
4320 REM -----------------------------------------------------------------
4330 :
4340 PRINT "■■"+LEFT$(D$,23)+LEFT$(BL$,39)+"□"
4350 A=INT(A)
4360 INPUT "WELLENFORM(0-255)";A
4370 IF A>255 THEN GOTO 4340
4380 :
4390 B=A
4400 NS(1)=27+2*S
4410 ST$(27+2*S)=STR$(A)
4420 GOSUB 7570
4430 S$=""
4440 :
4450 FOR I=8 TO 1 STEP-1
4460 : IF A(I)=1 THEN S$=S$+WL$(I)
4470 : IF A(I)=0 THEN S$=S$+"  0  "
4480 NEXT
4490 :
4500 NS(2)=28+2*S
4510 ST$(28+2*S)=S$
4520 ZS=2
4530 :
4540 NA(1)=7*S-3
4550 W(7*S-3)=A
4560 ZA=1
4570 :
4580 GOSUB7740
4590 :
4600 RETURN
4610 :
4700 REM -----------------------------------------------------------------
4710 REM ---                     ATTACK-ZEIT EINSTELLEN                    ---
4720 REM -----------------------------------------------------------------
4730 :
4740 PRINT "■■"+LEFT$(D$,23)+LEFT$(BL$,39)+"□"
4750 INPUT "ATTACK(0-15)";A
4760 IF A>15 OR A<0 THEN GOTO 4740
4770 :
4780 NS(1)=22+S
4790 ST$(22+S) = STR$((W(7*S-2) AND 15) OR A*16)
4800 ZS=1
4810 NA(1)=S*7-2
4820 W(S*7-2)=VAL(ST$(22+S))
4830 :
4840 GOSUB 7740
4850 :
4860 RETURN
4870 :
4900 REM -----------------------------------------------------------------
4910 REM ---                     DECAY-ZEIT EINSTELLEN                     ---
4920 REM -----------------------------------------------------------------
4930 :
4940 PRINT"■■"+LEFT$(D$,23)+LEFT$(BL$,39)+"□"
4950 INPUT "DECAY(0-15)";A
4960 IF A>15 OR A<0 THEN GOTO 4740
4970 :
4980 NS(1)=22+S
4990 ST$(22+S) = STR$((W(7*S-2) AND 15*16) OR A)
5000 ZS=1
5010 NA(1)=S*7-2
5020 W(S*7-2)=VAL(ST$(22+S))
5030 :
```

Listing 4/4.2.1-1 (Teil 6)

```
5040 GOSUB 7740
5050 :
5060 RETURN
5070 :
5100 REM ------------------------------------------------------------------
5110 REM ---                    SUSTEIN-LEVEL EINSTELLEN                 ---
5120 REM ------------------------------------------------------------------
5130 :
5140 PRINT "■■"+LEFT$(D$,23)+LEFT$(BL$,39)+"□"
5150 INPUT "SUSTAIN(0-15)";A
5160 IF A>15 OR A<0 THEN GOTO 4740
5170 :
5180 NS(1)=25+S
5190 ST$(25+S) = STR$((W(7*S-1) AND 15) OR A*16)
5200 ZS=1
5210 NA(1)=S*7-1
5220 W(S*7-1)=VAL(ST$(25+S))
5230 :
5240 GOSUB 7740
5250 :
5260 RETURN
5270 :
5300 REM ------------------------------------------------------------------
5310 REM ---                    RELEASE-ZEIT EINSTELLEN                  ---
5320 REM ------------------------------------------------------------------
5330 :
5340 PRINT "■■"+LEFT$(D$,23)+LEFT$(BL$,39)+"□"
5350 INPUT "RELEASE(0-15)";A
5360 IF A>15 OR A<0 THEN GOTO 4740
5370 :
5380 NS(1)=25+S
5390 ST$(25+S) = STR$((W(7*S-1) AND 15*16) OR A)
5400 ZS=1
5410 NA(1)=S*7-1
5420 W(S*7-1)=VAL(ST$(25+S))
5430 :
5440 GOSUB 7740
5450 :
5460 RETURN
5470 :
5500 REM ------------------------------------------------------------------
5510 REM ---                    LAUTSTAERKE EINSTELLEN                   ---
5520 REM ------------------------------------------------------------------
5530 :
5540 PRINT "■■"+LEFT$(D$,23)+LEFT$(BL$,39)+"□"
5550 INPUT "LAUTSTAERKE(0-15)";A
5560 :
5570 NS(1)=42
5580 ST$(42) = STR$((W(24) AND 15*16) OR A)
5590 :
5600 NS(2)=44
5610 ST$(44)=STR$(A)
5620 ZS=2
5630 :
5640 NA(1)=(24)
5650 W(24)=VAL(ST$(42))
5660 :
5670 GOSUB 7740
5680 :
5690 RETURN
5700 :
```

Listing 4/4.2.1-1 (Teil 7)

```
5800 REM ---------------------------------------------------------------------
5810 REM ---                          FILTER EINSTELLEN                    ---
5820 REM ---------------------------------------------------------------------
5830 :
5840 PRINT "■"+LEFT$(D$,23)+LEFT$(BL$,39)+"□"
5850 PRINT "F,R,A,Q?"
5860 GET A$
5870 :
5880 IF A$="" THEN GOTO  5860
5890 IF A$="F" THEN GOSUB  5980
5900 IF A$="R" THEN GOSUB  6300
5910 IF A$="A" THEN GOSUB  6490
5920 IF A$="Q" THEN GOSUB 6810
5930 :
5940 RETURN
5950 :
5960 REM ---                       FILTERECKFREQUENZ                       ---
5970 :
5980 PRINT "■"+LEFT$(D$,23)+LEFT$(BL$,39)+"□"
5990 INPUT "FILTERECKFREQUENZ(30-11902)";A
6000 IF A<=30 OR A>11902 THEN GOTO 5980
6010 :
6020 NS(1)=35
6030 ST$(35)=STR$(A)
6040 A=INT((A-30)/5.8)
6050 :
6060 NS(2)=36
6070 ST$(36)=STR$(A)
6080 L=A AND 7
6090 :
6100 NS(3)=37
6110 ST$(37)=STR$(L)
6120 H=(A AND 2040)/8
6130 :
6140 NS(4)=38
6150 ST$(38)=STR$(H)
6160 ZS=4
6170 :
6180 NA(1)=21
6190 W(21)=L
6200 NA(2)=22
6210 W(22)=H
6220 ZA=2
6230 :
6240 GOSUB 7740
6250 :
6260 RETURN
6270 :
6280 REM ---                       RESONANZ EINSTELLEN                     ---
6290 :
6300 PRINT "■"+LEFT$(D$,23)+LEFT$(BL$,39)+"□"
6310 INPUT "RESONANZ(0-15)";A
6320 IF A<0 OR A>15 THEN GOTO 6300
6330 :
6340 NS(1)=39
6350 ST$(39) = STR$((W(23) AND 15) OR A*16)
6360 NS(2)=40
6370 ST$(40)=STR$(A)
6380 ZS=2
6390 NA(1)=23
6400 W(23)=VAL(ST$(39))
6410 ZA=1
6420 :
6430 GOSUB 7740
6440 :
```

Listing 4/4.2.1-1 (Teil 8)

```
6450 RETURN
6460 :
6470 REM ---                    FILTERART EINSTELLEN                    ---
6480 :
6490 PRINT "■■"+LEFT$(D$,23)+LEFT$(BL$,39)+"■"
6500 INPUT "FILTERART(0-15)";A
6510 IF A<0 OR A>15 THEN GOTO 5840
6520 :
6530 NS(1)=42
6540 ST$(42) = STR$((W(24) AND 15) OR 16*A)
6550 B=A*16
6560 :
6570 GOSUB 7570
6580 :
6590 IF A(8)=1 THEN S$="3.AUS"
6600 IF A(8)=0 THEN S$="   0 "
6610 IF A(7)=1 THEN S$=S$+" HP"
6620 IF A(7)=0 THEN S$=S$+" 0 "
6630 IF A(6)=1 THEN S$=S$+" BP"
6640 IF A(6)=0 THEN S$=S$+" 0 "
6650 IF A(5)=1 THEN S$=S$+" TP"
6660 IF A(5)=0 THEN S$=S$+" 0 "
6670 :
6680 NS(2)=43
6690 ST$(43)=S$
6700 ZS=2
6710 NA(1)=24
6720 W(24)=VAL(ST$(42))
6730 ZA=1
6740 :
6750 GOSUB7740
6760 :
6770 RETURN
6780 :
6790 REM ---                    QUELLE FESTLEGEN                    ---
6800 :
6810 PRINT "■■"+LEFT$(D$,23)+LEFT$(BL$,39)+"■"
6820 INPUT "QUELLE(0-15)";A
6830 IF A<0 OR A>15 THEN GOTO 5840
6840 NS(1)=39
6850 ST$(39) = STR$((W(23) AND 16*7) OR A)
6860 B=A
6870 :
6880 GOSUB 7570
6890 :
6900 IF A(4)=1 THEN S$="EXT "
6910 IF A(4)=0 THEN S$=" 0  "
6920 IF A(3)=1 THEN S$=S$+"OS3 "
6930 IF A(3)=0 THEN S$=S$+" 0  "
6940 IF A(2)=1 THEN S$=S$+"OS1 "
6950 IF A(2)=0 THEN S$=S$+" 0  "
6960 IF A(1)=1 THEN S$=S$+"OS1 "
6970 IF A(1)=0 THEN S$=S$+" 0  "
6980 :
6990 NS(2)=41
7000 ST$(41)=S$
7010 ZS=2
7020 :
7030 NA(1)=23
7040 W(23)=VAL(ST$(39))
7050 ZA=1
7060 :
7070 GOSUB 7740
7080 :
```

Listing 4/4.2.1-1 (Teil 9)

```
7090 RETURN
7100 :
7200 REM -----------------------------------------------------------------
7210 REM ---                    STIMME AUSWAEHLEN                      ---
7220 REM -----------------------------------------------------------------
7230 :
7240 ST$(2)=" "
7250 ST$(3)=" "
7260 ST$(4)=" "
7270 ST$(VAL(A$)+1)="*"
7280 S=VAL(A$)
7290 :
7300 FOR I=1 TO 3
7310 : PRINT P$(1+I)ST$(1+I)
7320 NEXT
7330 :
7340 RETURN
7350 :
7400 REM -----------------------------------------------------------------
7410 REM ---                TON EIN-/ AUSSCHALTEN                      ---
7420 REM -----------------------------------------------------------------
7430 :
7440 IF T=0 THEN TT=1 : S$="TON EIN"
7450 IF T=1 THEN TT=0 : S$="TON AUS"
7460 :
7470 T=TT
7480 NS(1)=1
7490 ST$(1)=S$
7500 ZS=1
7510 ZA=0
7520 :
7530 GOSUB 7740
7540 :
7550 RETURN
7560 :
7570 FOR I=8 TO 1 STEP -1
7580 : A(I)=INT(B/(2↑(I-1)))
7590 : B=B-A(I)*2↑(I-1)
7600 NEXT
7610 :
7620 RETURN
7630 :
7700 REM -----------------------------------------------------------------
7710 REM ---                  TOENE AKTUALISIEREN                      ---
7720 REM -----------------------------------------------------------------
7730 :
7740 POKE WE(1),W(4)  AND 254
7750 POKE WE(2),W(11) AND 254
7760 POKE WE(3),W(18) AND 254
7770 :
7780 FOR I=1 TO ZA
7790 : POKE SI+NA(I),W(NA(I))
7800 NEXT
7810 :
7820 IF ZS=0 THEN GOTO 7880
7830 :
7840 FOR I=1 TO ZS
7850 : PRINT P$(NS(I))ST$(NS(I))
7860 NEXT
7870 :
7880 POKE WE(1),W(4)  AND (254 OR T)
7890 POKE WE(2),W(11) AND (254 OR T)
7900 POKE WE(3),W(18) AND (254 OR T)
7910 :
7920 RETURN
```

Listing 4/4.2.1-1 (Teil 10)

Programmbeschreibung

Wegen des großen Umfanges möchten wir Sie zu Beginn mit der groben Programmstruktur bekannt machen:

1000	Vorspann und Besetzen der Variablen	
2000	Hauptprogramm	
3000	Unterprogramme	
	3060	Informationstableau
	3350	Bildschirmgrundmaske
3600	Tonhöhe einstellen	
4000	Puls einstellen	
4300	Wellenform einstellen	
4700	Attack-Zeit einstellen	
4900	Decay-Zeit einstellen	
5100	Sustain-Level einstellen	
5300	Release-Zeit einstellen	
5500	Lautstärke einstellen	
5800	Filter einstellen	
	5960	Filtereckfrequenz
	6280	Resonanz einstellen
	6470	Filterart einstellen
	6790	Quelle feststellen
7200	Stimme auswählen	
7400	Ton ein-/ausschalten	
7700	Töne aktualisieren	

Wie man schon an der Programmübersicht sieht, ist für jede Registerart ein Unterprogramm zuständig. Beginnen wir jedoch die Besprechung des gesamten Programms mit dem Vorspann.

Vorspann

Im Vorspann werden zunächst einige Hilfsvariablen besetzt, die der Cursorsteuerung dienen. Anschließend werden die Felder zur Aufnahme der Registernummer und der Änderungsdaten definiert. In dem Feld WL$( ) werden die Texte zu den verschiedenen Wellenformen abgelegt, und in den Zeilen ab 1165 werden den entsprechenden Feldern die Registeradressen zugeordnet. Die Zeilen ab 1330 dienen der Cursorpositionierung für die einzelnen Elemente der Bildschirmmaske.

Hauptprogramm

Das Hauptprogramm besteht nur aus dem Anzeigen des Informationstableaus und dem Sprungverteiler in die verschiedenen Unterprogramme aufgrund einer einbuchstabigen Eingabe. Sie haben folgende Wahlmöglichkeiten:

Eingabe	Funktion
I	Informationstableau
T	Tonhöhe einstellen
P	Pulsfrequenz einstellen
W	Wellenform einstellen
A	Attack-Zeit einstellen
D	Decay-Zeit einstellen
S	Sustain-Level einstellen
R	Release-Zeit einstellen
F	Filter auswählen
L	Lautstärke einstellen
1	1. Stimme auswählen
2	2. Stimme auswählen
3	3. Stimme auswählen
' '	Ton ein-/ausschalten (Leertaste)

Unterprogramm Informationstableau

Um Ihnen die Arbeit am Rechner — ohne dauerndes Nachschauen in einer Bedienungsanleitung — zu erleichtern, wurde ein Informationstableau in das Programm eingebaut, das alle Eingaben anzeigt.

Das Informationstableau verlassen Sie durch Drücken der Taste „I", wonach das Programm ab Zeile 3350 fortgeführt und die Grundbildschirmmaske ausgegeben wird.

Allgemeines zu den Unterprogrammen zum Einstellen der Register

Die Unterprogramme zum Einstellen der Register arbeiten alle nach dem gleichen Prinzip. Zunächst wird der Cursor entsprechend innerhalb der Bildschirmmaske positioniert und anschließend der neue Wert erfragt. Dieser wird noch auf Plausibilität geprüft. Anschließend werden alle nötigen Änderungen in den Feldern ST$( ) — für die Bildschirmausgabe — und W( ) — für die Register — durchgeführt. Die Anzahl der Änderungen im Feld ST$( ) wird in der Variablen ZS (Zahl der Stringänderungen) festgehalten und die Anzahl der Änderungen für die Register in der Variablen ZA (Zahl der Aenderungen).

Dann werden noch zwei Felder NS( ) und NA( ) mit den Nummern der Zeichenreihen und Änderungen besetzt. NS( ) und NA( ) bestimmen also, welche Zeichenreihen und Werte geändert wurden.

Zum Abschluß wird das Unterprogramm ab Zeile 7700 aufgerufen, mit dem die Töne aktualisiert werden.

Im folgenden wollen wir nur noch kurz auf die Besonderheiten der einzelnen Unterprogramme eingehen.

Unterprogramm Tonhöhe einstellen

Beim Einstellen der Tonhöhe muß berücksichtigt werden, daß die Frequenz in zwei verschiedenen Bytes abgespeichert wird (L/Lower Byte; H/Higher Byte). Außerdem muß noch die eingegebene Frequenz mit einem Faktor multipliziert werden, der den Computer veranlaßt, die entsprechende Tonhöhe auszugeben.

Unterprogramm Puls einstellen

Das Einstellen des Pulses geschieht genauso in zwei Byte wie das Einstellen der Frequenz (L/H).

Unterprogramm Wellenform einstellen

Beim Einstellen der Wellenform wird noch am Bildschirm in Worten angezeigt, welche Wellenform eingestellt wurde (vergleiche ab Zeile 1110).

Unterprogramm Attack-Zeit einstellen

Die Attack-Zeit wird in den jeweiligen Registern in den höherwertigen vier Bit eingestellt, so daß die Eingabe mit 16 multipliziert werden muß.

Unterprogramm Decay-Zeit einstellen

Die Decay-Zeit wird in den gleichen Registern wie die Attack-Zeit eingestellt, jedoch sind hier die niederwertigen vier Bit zuständig. Sowohl bei dem Attack-Zeit einstellen als auch beim Decay-Zeit einstellen müssen die jeweils anderen Werte natürlich unverändert bleiben.

Unterprogramm Sustain-Level einstellen

Ebenso wie bei Attack/Decay ist die Aufteilung bei dem Sustain-Level der Release-Zeit. In den entsprechenden Registern sind die vier höherwertigen Bit für den Sustain-Level zuständig. Die niederwertigen vier Bit dürfen natürlich nicht geändert werden.

Unterprogramm Release-Zeit einstellen

Wie schon erwähnt geschieht das Einstellen der Release-Zeit in den gleichen Registern, wo auch der Sustain-Level eingestellt wird. Angesprochen werden die niederwertigen vier Bit.

Unterprogramm Lautstärke einstellen

Die Lautstärke wird insgesamt für alle drei Stimmen eingestellt und hat ein ganzes Register zur Verfügung, obwohl die Lautstärke nur Werte zwischen 0 und 15 annehmen kann.

Unterprogramm Filter einstellen

Bei der Einstellung der Filter (wo auch ein Register für alle drei Stimmen zuständig ist) wird zunächst in einem kurzen Menü gefragt, welche Parameter für die Filtereinstellung geändert werden sollen. Wie in der Programmübersicht aufgezeigt, können geändert werden:

- Filtereckfrequenz
- Resonanz
- Filterart
- Quelle

Unterprogramm Stimme auswählen

Beim Auswählen der Stimme wird die Variable S als Merker für die anderen Programmteile besetzt. Außerdem wird noch ein „*" bei der jeweiligen angewählten Stimme am Bildschirm angezeigt.

Unterprogramm Ton ein-/ausschalten

Das Ein- und Ausschalten des Tones geschieht über das sogenannte Key-Bit. An dieser Stelle sei angemerkt, daß Sie bei eingeschalteter Release-Zeit auch den Ton öfters ein-/ausschalten sollten, da Sie dann erst den richtigen Klang für die Release-Zeit erhalten. Wie aus dem Bild in diesem Kapitel ersichtlich, beginnt die Release-Zeit erst, nachdem das Key-Bit ausgeschaltet wurde.

Unterprogramm Töne aktualisieren

Im letzten Unterprogramm — das von allen Änderungsunterprogrammen aufgerufen wird — werden alle eingegebenen Änderungen durchgeführt. Hierzu werden — wie vorher beschrieben — die Variablen ZA und ZS herangezogen, die ja jeweils die Zahl der Änderungen angeben.

Das Programmm befindet sich auf der Grundwerksdiskette

Variablenübersicht

Variable	gültiger Bereich	Bedeutung
A	0 . . . 9	Variable zur Eingabe einer Ziffer
A$	bel. Zeichen	Variable zur Eingabe eines Zeichens
B	0 . . . 255	Übergabeparameter für Bestimmung der Bitkombination eines Byte
BL$	Leerzeichen	Kette aus Leerzeichen
CL$	Cursor links	Steuerzeichenreihe
CD$	Cursor nach unten	Steuerzeichenreihe
H	0 . . . 255	Wert für Frequenz High Byte
I	0 . . . 44	Schleifenvariable
IH	54294	Register Filterfrequenz High Byte
IL	54293	Register Filterfrequenz Low Byte
L	0 . . . 255	Wert für Frequenz Low Byte
LA	54296	Register Lautstärke
P	0 . . . 65535	Parameter für Frequenz
R$	Cursor rechts	Steuerzeichen
RF	54295	Register Resonanz/Filter ein
S	1, 2, 3	Stimme
S$	bel. Zeichen	Variable zum Zusammensetzen von Ausgabe-zeichenreihen
SI	54272	Register Basisadresse
T	0, 1	Merker für Ton ein/Ton aus
TT	0, 1	Hilfsvariable für Umschalten Ton ein/aus, aus/ein
ZA	0, 1, 2	Zahl der Änderungen der Registerwerte

Felder (Arrays)

Name	Dimen.	Bereich	Bedeutung
A( )	1 . . . B	0, 1	Enthält die Bitkombination von B (A(1)=1: 1. Bit gesetzt)
AD( )	1 . . . 3	54272 . . . 54296	Register Attack/Decay
FH( )	1 . . . 3	54272 . . . 54296	Register Frequenz High Byte
FL( )	1 . . . 3	54272 . . . 54296	Register Frequenz Low Byte
NA( )	1 . . . 3	0 . . . 24	Nummer der Änderung, die gemacht werden soll
NS( )	1 . . . 10	1 . . . 44	Nummer der Strings, die ausgegeben werden sollen
P$	1 . . . 44	Steuerzeichen	Gibt die Position der Ausgabestrings an
PH( )	1 . . . 3	54272 . . . 54296	Register Puls High Byte
PL( )	1 . . . 3	54272 . . . 54296	Register Puls Low Byte
SR( )	1 . . . 3	54272 . . . 54296	Register Sustain/Release
ST$( )	1 . . . 44	alle Zeichen	String für Bildschirmausgabe
W( )	1 . . . 24	0 . . . 255	Werte aller Musik-Register
WE( )	1 . . . 3	54272 . . . 54296	Register Wellenform
WL$( )	1 . . . 8	alle Zeichen	Abkürzungen der Bedeutung der einzelnen Bits der Wellenform

4/4.3

Grafik beim C 128 PC

4/4.3.1

Hochauflösende Grafik

Wie die meisten anderen Home-Computer auch, verfügt der C 128 über hervorragende Grafikeigenschaften, insbesondere ist hierbei die große Zahl von Grafikbefehlen zu nennen, die kaum noch Wünsche offen lassen und um einiges mehr bieten, als bei den meisten Home-Computern anzutreffen ist. Mit seinem direkten Vorgänger — dem C 64 — hat der C 128 zwar den grundlegenden Grafikablauf noch gemeinsam, jedoch gibt es einige Grafikbefehle, die beim C 64 noch nicht einmal mit umständlicher POKEerei zu realisieren waren, sondern sogar Machinenspracheprogramme erfordert hätten.

Das Kapitel über hochauflösende Grafik werden wir wegen der Vielzahl der Befehle untergliedern, wobei wir jeweils anhand von Beispielen die Wirkungsweise der Befehle besprechen wollen. Zunächst wollen wir uns natürlich den allgemeinen Grafikbefehlen zuwenden und die Befehle am Beispiel eines Zeichenprogrammes mit dem Joystick besprechen.

Den Figuren zum Zeichnen einer Linie, eines Rechteckes und eines Kreises ist ein eigenes Kapitel gewidmet, wobei wir hier einige kleine künstlerische Grafiken erstellen wollen. Die allgemeinen Grafikbefehle werden wir mit der Vorstellung eines kleinen Programmes beenden, das uns die aktuellen Daten des Grafikcursors, der Farbe und einiges andere in gesammelter Form ausgibt.

4/4.3.1.1

Allgemeine Grafikbefehle

Wie bereits erwähnt, wollen wir die allgemeinen Befehle zum Thema hochauflösende Grafik anhand eines Zeichenprogrammes darstellen, wobei der Zeichenstift durch den Joystick geführt werden soll. Dazu ist es sinnvoll, das Kapitel über Joystick (2/3.5) vorher unter die Lupe zu nehmen.

Wir haben dieses Beispiel gewählt, da wir damit in der Lage sind, fast alle Befehle im Zusammenhang zu besprechen, und Ihnen außerdem sofort ein fertiges Programm zur Verfügung steht, mit dem Sie Grafiken nach eigenen Wünschen entwerfen können. Selbstverständlich läßt sich das Programm noch Ihren Bedürfnissen anpassen und auch ergänzen.

Im Rahmen der nachfolgenden Teilkapitel werden wir sukzessive das Zeichenprogramm entwickeln. Abschließend wird nochmal das gesamte Listing ausgedruckt, sowie eine Variablenübersicht aufgeführt, um Ihnen Änderungen und Ergänzungen anhand des bestehenden Listings zu vereinfachen. Für „Nur-Anwender“ ist auch eine Bedienungsanleitung beigefügt.

4/4.3.1.1.1

GRAPHIC — auf geht's

Zu Beginn eines jeden Grafik-Programmes muß die GRAPHIC-Anweisung gegeben werden, mit der Sie auswählen können, welchen Grafik-Modus Sie einstellen.

Zunächst etwas Grundsätzliches zum GRAPHIC-Befehl. Durch die verschiedenen Modi (0-5) haben Sie sowohl eine Wahl zu treffen zwischen hochauflösender Grafik und Mehrfarben-Grafik als auch zwischen normalen Textbildschirmen, sowie eine Mischung von beiden. Werden Text und Grafik gemischt, so befinden sich die Textzeilen immer am unteren Bildschirmrand. Besonders diese beiden Modi bieten einen entscheidenden Vorteil, da selten reine Grafikanwendungen vorkommen, meist

jedoch auch noch eine Art Menüsteuerung — wie in unserem Beispiel — vorhanden ist.

Für unser Beispielprogramm verwenden wir Mehrfarben-Grafik mit zusätzlichem Text (Modus: 4), wobei unser Textteil in der 20. Zeile beginnen soll. Außerdem soll natürlich der Grafik-Bildschirm zu Beginn von eventuell vorhandenen Zeichnungen befreit werden, womit unsere Eingabe wie folgt aussehen muß:

```
1000 REM -------------------------------------------
1010 REM ---  ZEICHENPROGRAMM  ---
1020 REM -------------------------------------------
1030 :
1040 GRAPHIC 4, 1, 20
1050 :
```

Der Befehl GRAPHIC CLR dient allerdings nicht — wie vielleicht vermutet — zum Löschen des Grafikanteiles auf dem Bildschirm, sondern gibt lediglich den reservierten Speicherbereich am Beginn des BASIC-Programmspeichers frei. Wie man doch eine Grafik löschen kann, zeigen wir Ihnen im Kapitel Tips und Tricks.

Der Modus 5 beim GRAPHIC-Befehl funktioniert natürlich nicht bei einem 40-Zeichen-Bildschirm. Wenn Sie GRAPHIC 5 eingeben, so wechselt der Cursor vom 40-Zeichen-Bildschirm auf den 80-Zeichen-Bildschirm, hat also die gleiche Auswirkung wie ESC X. Die Wirkung von ESC X in umgekehrter Richtung erhalten Sie durch GRAPHIC 0.

Sofern Sie Ihre Funktionstasten nicht anders belegt haben, steht Ihnen der GRAPHIC-Befehl nach Drücken der Funktionstaste F1 zur Verfügung.

4/4.3.1.1.2

COLOR — wir färben ein

Der nächste Schritt in einem Grafikprogramm sollte sich mit dem Festlegen der benötigten Farben beschäftigen. Diese Wahl ist jedoch nicht endgültig, da später im Programm beliebig umgefärbt werden kann.

Dabei sollte man jedoch die Vorgehensweise bei der Grafikdarstellung berücksichtigen, wo im Farbspeicher durch Bits jeweils entweder die Hintergrund- oder Vordergrundfarbe bei der hochauflösenden Grafik ausgewählt wird, bzw. beim Mehrfarben-Modus zusätzlich die beiden Zusatzfarben 1 oder 2. Im Bildschirmspeicher befindet sich also nicht eine der Farbnummern 1 bis 16, sondern nur eine der Nummern 0 oder 1 (Bit im Farb-RAM) bei hochauflösender Grafik bzw. 0 bis 3 (dargestellt in zwei Bits) beim Mehrfarben-Modus.

Für unser Beispielprogramm wählen wir für den Hintergrund die Farbe gelb (8), für den Vordergrund die Farbe blau (7) und für die beiden Farben im Mehrfarben-Modus dunkelgrün (6) und hellbraun (9).

Die Randfarbe passen wir dem Hintergrund an, so daß zu Beginn der gesamte Bildschirm in gelb erscheint.

Da das Farb-RAM des Grafik-Modus und das Farb-RAM des Text-Modus im Rechner unterschiedlich gehandhabt werden, ist es auch möglich, dem Text eine grundsätzlich andere Farbe zu geben, als bei den Grafikfarben vorgewählt wurde. Wir wählen aus Kontrastgründen für unsere Textfarbe also schwarz (1).

Unser Programm würde mit obigen Vorgaben also wie folgt weitergeführt:

```
1060 COLOR 0,8
1070 COLOR 1,7
1080 COLOR 2,6
1090 COLOR 3,9
1100 COLOR 4,8
1110 COLOR 5,1
1120 :
```

Umsteiger vom C 64 sollten auf jeden Fall beachten, daß alle Farbcodes jeweils eine Ziffer höher ausfallen, schwarz also nicht mehr durch die Ziffer 0 dargestellt wird, sondern durch die Ziffer 1. Die gesamte Palette reicht also nicht mehr von 0 bis 15, sondern von 1 bis 16.

4/4.3.1.1.3

DRAW — Rahmen zeichnen

Obwohl wir im Kapitel über grafische Figuren den DRAW-Befehl nochmals besprechen werden, wollen wir ihn an dieser Stelle dazu heranziehen, einen Rahmen um den Grafikbereich des Bildschirms zu ziehen. Dabei nutzen wir eine bemerkenswerte Eigenschaft des DRAW-Befehls aus, da wir einen Linienzug aus mehreren einzelnen Linien mit einem einzigen Befehl realisieren können.

Ein Befehl zum Zeichnen eines einzigen Punktes liegt im Basic 7.0 des C 128 nicht vor. Auch für die Darstellung eines Punktes ist also der DRAW-Befehl heranzuziehen. Wenn man sich dazu die Bemerkungen im Handbuch etwas genauer durchliest, wird man feststellen, daß zum Zeichnen eines Punktes an der Position des Grafikcursors lediglich DRAW (ohne jegliche Parameter) eingegeben werden muß. Dies gilt natürlich nur, sofern die Ausgabe in der aktuellen Farbquelle gewünscht ist. Laut Bemerkung im Handbuch wird ein Punkt gezeichnet, wenn Start- und Endkoordinaten gleich sind, aber die Startkoordinate braucht nicht angegeben zu werden, wenn die Position des Grafik-Cursors gemeint ist.

Welche Punkte am Bildschirm müssen nun für unseren Rahmen verbunden werden? Beginnen wollen wir an der oberen linken Bildschirmecke, die — entgegen dem karthesischen Koordinatensystem — die Position 0,0 aufweist. Da wir im Mehrfarben-Modus arbeiten, hat die äußerste X-Koordinate die Position 159. Aufgrund unserer Textzeilen am unteren Rand des Bildschirmes müssen wir von der maximal üblichen Y-Ausdehnung (199) noch 5x8=40 mögliche Koordinaten abziehen, wodurch wir auch als größtmögliche Y-Ausdehnung im Grafikbereich 159 erhalten. Die Fortsetzung in unserem Programm sieht also wie folgt aus:

```
1130 DRAW 1,0,0 TO 0,159 TO 159,159 TO 159,0 TO 0,0
1140 :
```

Auch den Textbereich wollen wir einrahmen, wofür wir allerdings nicht die hochauflösende Grafik heranziehen können. Hier verwenden wir die in Kapitel 4/4.1.1 beschriebenen Grafikzeichen wie folgt:

```
1150 PRINT"┌──────────────────────────────────────┐";
1160 PRINT"│                                      │";
1170 PRINT"│                                      │";
1180 PRINT"│                                      │";
1190 PRINT"└──────────────────────────────────────┘";
1200 :
```

Damit haben wir die Grundlagen für weitere Ausgaben am Bildschirm durchgeführt und können uns nun dem Menü zuwenden, wie es folgende Zeilen zeigen:

```
2000 REM ------------------------------
2010 REM ---   MENUE  UND ZEIGER     ---
2020 REM ------------------------------
2030 :
2040 PRINT"▮QQQQQQQQQQQQQQQQQQQQQQQ";
2050 PRINT"▮Z-ZEICHNEN  M-MERKEN   S-STIFT       |";
2060 PRINT"▮B-BOX       P-AUSFUEL. W-PINSEL      |";
2070 PRINT"▮K-KOPIEREN  F-FARBE    U-UEBERNEHMEN |";
2080 :
```

Da wir möglichst viel Platz auf dem Bildschirm für den Grafikbereich vorsehen wollten, haben wir das Menü kurz gehalten. Sie können das Menü jedoch unter Änderung von Zeile 1040 auch auf einen größeren Bildschirmbereich ausdehnen und zusätzliche Hinweise anfügen.

Weil die HOME-Position des Textcursors sich jedoch immer noch in der linken oberen Bildschirmecke befindet, und nicht — wie vielleicht vermutet — in der linken oberen Ecke des Textfensters, muß zunächst der Cursor auf den Anfang des Textbereiches positioniert werden, was Zeile 2040 erledigt. Damit wir den Rahmen aus den Zeilen 1150 bis 1190 nicht zerstören, wird das erste Zeichen einer Textzeile jeweils mit einem „Cursor nach rechts“ übersprungen. Eine andere Form haben wir zum Ende einer Menüzeile gewählt, wo wir das Grafikzeichen erneut ausgegeben haben.

Die einzelnen Punkte des Menüs werden wir im Folgenden jeweils bei der Tastaturabfrage und den entsprechenden Unterprogrammen besprechen.

Bevor wir nun im Programmlisting weitergehen, sollten Sie zunächst einen kleinen Probelauf durchführen und anschließend das Programm sichern.

Ehe wir die Joystickabfrage, die Tastaturabfrage und die einzelnen Unterprogramme besprechen, müssen wir zunächst eine kleine Exkursion zu den Sprites machen.

4/4.3.1.1.4

Sprites und Grafik

Warum hier Sprites verwenden? Sprites kann man als freibewegbare Grafiksymbole umschreiben, die entweder vor oder hinter einer grafischen Darstellung am Bild-

schirm positioniert werden können. Diese Eigenschaft wollen wir uns für die Darstellung eines Cursors und die Merkposition zunutze machen.

Da wir für ein Zeichenprogramm mit dem Joystick aus Gründen der Bedienungsfreundlichkeit jeweils angeben müssen, wo sich der Grafikcursor aktuell befindet, sollte ein Cursorzeichen an der entsprechenden Position am Bildschirm positioniert werden. Bei Verwendung einer Ansammlung von Bildschirmpunkten ist dies recht umständlich, da sie zunächst an eine Position gebracht werden müssen, um bei Verschieben des Grafikcursors zunächst an der alten Position gelöscht und an der neuen Position gezeichnet werden müssen. Sicherlich wäre dies mit den Befehlen SSHAPE und GSHAPE relativ einfach möglich (im Gegensatz zur ausschließlichen Verwendung des DRAW-Befehls), jedoch bieten die Sprites hier einige Vorteile, insbesondere die Priorität gegenüber der Zeichnung.

Zunächst sollten Sie mittels des SPRDEF-Befehls zwei Sprites definieren, die in etwa die Gestalt haben, wie sie in den Bildern 4/4.3.1-1 (Cursor) und 4/5.3.1-2 (Merker) abgebildet sind.

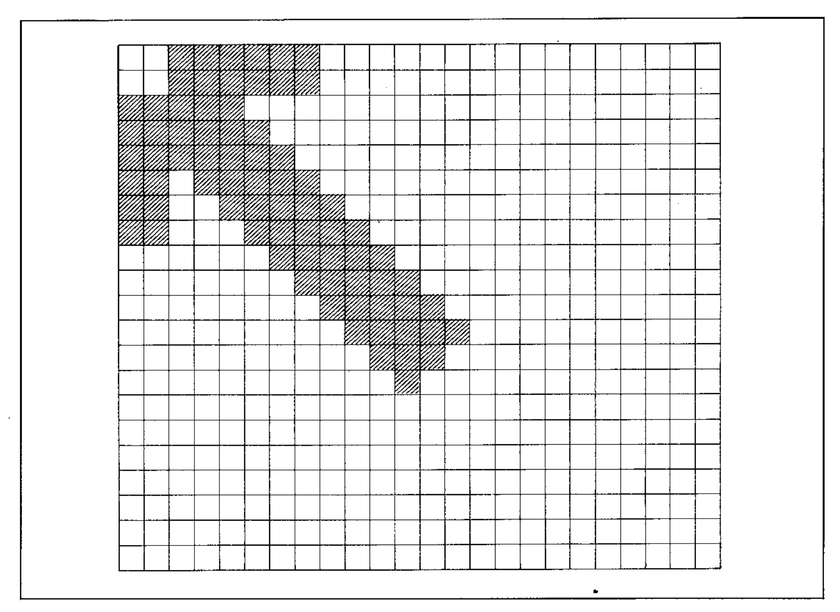


Bild 4/3.1-1 Zeiger auf aktuelle Cursorposition

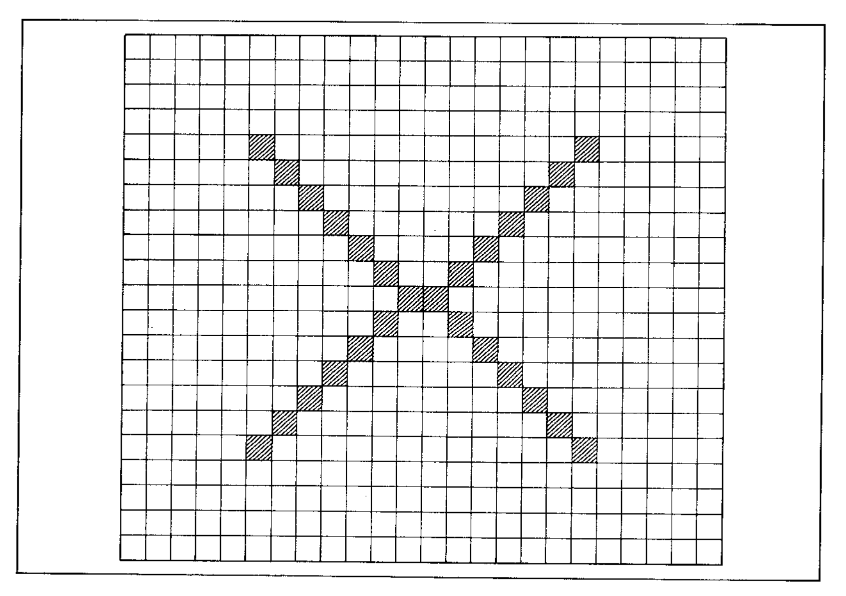


Bild 4/4.3.1-2 Zeiger für gemerkte Position

Die obere linke Ecke des Pfeiles aus Bild 4/4.3.1-1 (hiervon die obere Hälfte) stellt die Position des Grafikcursors dar, wobei durch die untere Hälfte in der Pfeilspitze noch eine Unterscheidung zum Pfeil gegeben ist.

Das „X“ in Bild 4/4.3.1-2 ist absolut symmetrisch (auch innerhalb des Sprites) angeordnet und bedeckt bei dem von uns gewählten Mehrfarben-Modus im Schnittpunkt genau die zu markierende Stelle.

Beide Sprites sollten Sie mit dem BSAVE-Kommando, das im Handbuch angegeben ist, auf Diskette ablegen, und zwar unter dem Namen „ZEICHSPRITE“.

Nachdem die Sprites definiert sind, sollten Sie das zuvor begonnene Programm wieder von Diskette laden. Da nicht immer davon ausgegangen werden kann, daß sich die Sprites im Hauptspeicher befinden (wie momentan direkt nach der Erstellung), setzen wir unser Programm fort, indem wir zunächst die Daten der Sprites von der Floppy in den Hauptspeicher übernehmen.

```
2090 REM BLOAD,,ZEICHSPRITE"
2100 :
```

Als nächstes muß das Sprite natürlich am Bildschirm sichtbar gemacht und an der richtigen Position dargestellt werden. Dazu dienen uns die Befehle SPRITE und MOVSPR.

Beim SPRITE-Befehl wählen wir neben der Nummer und dem Einschaltzustand die Farbe schwarz, korrespondierend zu unserer Textfarbe. Dargestellt werden soll das Sprite vor vorhandenen Zeichnungen, was durch die Priorität 0 ausgedrückt wird. Eine Ausdehnung in Y- bzw. X-Richtung ist nicht vorgesehen, kann von Ihnen aber leicht nachträglich eingebaut werden.

Obwohl wir für unsere Grafik den Mehrfarben-Modus gewählt haben, können wir das Sprite im normalen hochauflösenden Modus darstellen. Bei der Positionierung gibt es allerdings einige Schwierigkeiten, da die Koordinaten von Sprites anderen Gesetzen unterliegen, wie die Grafik-Koordinaten. Zu Beginn wollen wir den Grafik-Cursor in der Mitte des Grafikteils am Bildschirm positionieren. Die Grafik-Koordinaten sind also 80, 60, wie wir später noch sehen werden.

Wie Sie vielleicht schon beim Ausprobieren von Sprites festgestellt haben, können die Sprites hinter den Bildschirmrändern verschwinden. Um dies zu realisieren muß also gegenüber der Grafik-X-Koordinate 0 eine Koordinate 0 für Sprites existieren, die hiervon um -24 abweicht. Außerdem ist zu berücksichtigen, daß wir unsere Grafik-X-Koordinaten aufgrund des Mehrfarben-Modus errechnen, Sprite-Koordinaten diese Rechenweise jedoch nicht bekannt ist. Zur Positionierung eines Sprites in X-Richtung müssen wir also die aktuelle Grafix-X-Koordinate mit zwei multiplizieren und 24 addieren, was bei der anfänglichen Positionierung den Wert von 184 ergibt.

Die Multiplikation mit zwei fällt natürlich bei der Y-Koordinate weg. Hier ist jedoch ein Unterschied zwischen der Sprite-0-Koordinate und Grafik-0-Koordinate von 50 festzustellen, wie Sie durch Probieren leicht nachvollziehen können. Die Y-Koordinate unseres Sprite-Cursors muß also den Wert 110 erhalten, wodurch folgende Zeilen als Fortsetzung unseres Programmes eingegeben werden müssen:

```
2110 SPRITE 1,1,1,0,0,0,0
2120 SPRITE MOVSPR 1,84,110
2130 :
```

4/4.3.1.1.5

LOCATE — Grafik-Cursor positionieren

Zum LOCATE-Befehl gibt es nicht viel zu sagen. Sie sollten jedoch immer bedenken, daß sich der Grafik-Cursor nach einer Ausgabe immer an der zuletzt gezeichneten Position befindet. Leider ist der Befehl für den Text-Cursor nicht anwendbar, was einige umständliche Transaktionen (siehe Zeile 2040) ersparen würde.

Wie bereits erwähnt, soll der Grafik-Cursor zu Beginn die Position 80, 60 einnehmen, was wir durch den Befehl

```
2140 LOCATE 80, 60
2150 :
```

erreichen. Außerdem sind noch einige andere Variablen vorzubesetzen, u.a. die beiden Variablen, in denen wir uns die aktuelle Position des Grafik-Cursors merken, um diese nicht jeweils mit dem Befehl RDOT — feststellen zu müssen.

Ebenso sollten wir in einer Variablen F die aktuelle Zeichenfarbe (Farbquelle) ablegen und in der Variablen W (WIDTH) die Strichstärke für unseren Pinsel. Beide Variablen sollen mit „1“ voreingestellt sein, wonach die letzten Zeilen im Vorspann wie folgt einzugeben sind:

```
2160 AX=80
2170 AY=60
2180 :
2190 F=1
2200 :
2210 W=1
2220 :
```

4/4.3.1.1.6

Joystickabfrage

Bevor wir nun weitere grafische Befehle kennenlernen, wollenwir zuerst die Joystickabfrage programmäßig behandeln. Hierzu sei die Lektüre des Kapitels über Joystick empfohlen. Auf jeden Fall wählen wir für unsere zeichnerischen Tätigkeiten den Joystick-Port 2 (= Control-Port), um Kollisionen mit der Tastatur zu verhindern. Würden sie den Control-Port 1 wählen, so würde bei jedem Druck auf die Feuertaste das Zeichen „M" in den Tastaturpuffer wandern und weiter unten bei der Tastaturabfrage entsprechend erkannt werden. Dort würde dann ein Unterprogramm aufgerufen, das gar nicht gewünscht wurde.

Bei der Joystickabfrage wollen wir den aktuell abgefragten Wert in der Variablen JO ablegen, um bei Änderungen des Wertes in der weiteren Programmbehandlung keine Komplikationen entstehen zu lassen. Generell wird vor der Abfrage der Richtung geprüft, ob der Feuerknopf gedrückt wurde. In diesem Falle wird eine Variable TA mit „1" besetzt, um diesen Umstand später nutzen zu können. In diesem Fall wird außerdem vom eingelesenen Joystickwert der Wert für den Feuerknopf (128) subtrahiert, um für die Richtungen nur noch zwischen acht Möglichkeiten wählen zu können. Die gesamte Joystickabfrage auf einen Blick:

```
3000 REM -----------------------------
3010 REM ---    JOYSTICKABFRAGE     ---
3020 REM -----------------------------
3030 :
3040 JO=JOY(2)
3050 :
3060 IF JO>127 THEN TA=1 : JO=JO-128
3070 :
3080 IF JO=1 THEN AY=AY-1 : GOTO 3170
3090 IF JO=2 THEN AX=AX+1 : AY=AY-1 : GOTO 3170
3100 IF JO=3 THEN AX=AX+1 : GOTO 3170
3110 IF JO=4 THEN AX=AX+1 : AY=AY+1 : GOTO 3170
3120 IF JO=5 THEN AY=AY+1 : GOTO 3170
3130 IF JO=6 THEN AX=AX-1 : AY=AY+1 : GOTO 3170
3140 IF JO=7 THEN AX=AX-1 : GOTO 3170
3150 IF JO=8 THEN AX=AX-1 : AY=AY-1 : GOTO 3170
3160 :
3170 MOVSPR 1,AX*2+24,AY+50
3180 :
3190 IF TA THEN TA=0 : IF ZU$="Z" THEN DRAW F,AX,AY
3200 :
3210 IF AX<0   THEN AX=0
3220 IF AX>159 THEN AX=159
3230 IF AY<0   THEN AY=0
3240 IF AY>199 THEN AY=199
3250 :
```

```
3260 PRINT"S■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■";
3270 PRINT"■        ■■■■■■■X:";AX
3280 PRINT"S■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■";
3290 PRINT"■        ■■■■■■■Y:";AY
3300 :
```

Nach den schon beschriebenen Zeilen 3040 und 3060 werden die acht möglichen Richtungen für den Joystick abgeprüft und entsprechend unsere Merker für die Grafikposition geändert (AX, AY). Wurde eine Richtung erkannt, so wird direkt zur weiteren Behandlung übergegangen, was einigen Rechenzeitvorteil bringt.

In Zeile 3170 wird Sprite 1 neu positioniert, wobei die im vorletzten Kapitel beschriebene Rechenweise benutzt wird.

Dann wird abgefragt, ob bei Übernahme des Joystickwerts der Feuerknopf gedrückt war. In diesem Falle wird der Merker TA zurückgesetzt und — wenn der Zustand „ZEICHNEN" vorliegt — ein Punkt am Bildschirm dargestellt.

Um Fehler und einen daraus resultierenden Abbruch des Programmes zu verhindern, wird in den Zeilen 3210 bis 3240 die aktuelle Position des Grafik-Cursors überprüft, um ihn gegebenenfalls in seine Schranken (Grafik-Bildschirm) zurückzuweisen.

Als Hilfestellung für den Anwender dienen die Zeilen 3260 bis 3290, die in der rechten oberen Ecke des Textfensters die aktuelle Position in Ziffern ausgeben. Dies ist besonders für das exakte Zeichnen wichtig.

4/4.3.1.1.7

Tastaturabfrage

Nachdem aufgrund des eingelesenen Joystickwertes entsprechend verfahren wurde, muß als nächstes die Tastatur auf eine Eingabe abgefragt werden, was folgendes Programmstück realisiert:

```
4000 REM -----------------------------------
4010 REM ---   TASTATURABFRAGE       ---
4020 REM -----------------------------------
4030 :
4040 GET A$
4050 IF A$<>"" THEN B$=A$ : A$="": ELSE GOTO 3040
4060 :
4070 IF B$="Z" THEN GOSUB 10000 : GOTO 3040
4080 IF B$="P" THEN GOSUB 11000 : GOTO 3040
4090 IF B$="F" THEN GOSUB 12000 : GOTO 3040
4100 IF B$="S" THEN GOSUB 13000 : GOTO 3040
4110 IF B$="W" THEN GOSUB 14000 : GOTO 3040
4120 IF B$="M" THEN GOSUB 15000 : GOTO 3040
4130 IF B$="B" THEN GOSUB 16000 : GOTO 3040
4140 IF B$="U" THEN GOSUB 17000 : GOTO 3040
4150 IF B$="K" THEN GOSUB 18000 : GOTO 3040
4160 IF B$="L" THEN RUN
4170 :
4180 B$=""
4190 :
4200 GOTO 3040
4210 :
```

Der GETKEY-Befehl kann an dieser Stelle nicht zum Einsatz gebracht werden, da er wartet, bis eine Taste gedrückt wurde. Bei normalen Bewegungen des Grafik-Cursors mit Hilfe des Joysticks würde dies zu unnötigen Komplikationen führen, da jeweils für einen Schritt Bewegung auch eine Taste gedrückt werden muß. Wir beschreiten also den alten Weg mit dem GET-Befehl und fragen anschließend nach, ob ein Tastendruck vorlag. Wenn nicht, kann sofort zur erneuten Abfrage des Joystickwertes übergegangen werden, was wiederum Rechenzeitvorteile bringt.

Wurde eine Taste gedrückt, so wird deren Wert (Zeichen) der Variablen B$ übergeben, aufgrund deren Inhalt ab Zeile 4070 verzweigt wird.

Der Programmverteiler entspricht weitgehend den mnemotechnischen Bezeichnungen. Die dazu benötigten Unterprogramme sind in 1000-Schritten ab Zeile 10000 untergebracht, so daß Sie ab Zeile 19000 weitere Unterprogramme hinzufügen können.

Nachdem ein Unterprogramm abgehandelt wurde, wird wieder in den Menübereich gesprungen. Dies haben wir realisiert, damit eventuelle Programmänderungen einfacher durchgeführt werden können.

Etwas Rechenzeit würde gespart, wenn die einzelnen Programmabschnitte ab Zeile 10000 selbst jeweils zu Zeile 3040 übergehen würden.

Das vorliegende Menü läßt folgende Tätigkeiten zu:

Z	:	Zeichnen
P	:	Ausfüllen (PAINT)
F	:	Farbe ändern
S	:	Stiftfarbe ändern
W	:	Pinselstärke ändern (alternierend/WIDTH)

M	: Aktuelle Position merken
B	: Rechteck zeichnen (BOX)
U	: Grafik als Daten übernehmen (SSHAPE)
K	: Grafikbereich kopieren (GSHAPE)
L	: Grafik löschen (durch Neustart)

Wurde keine der vorgenannten Tasten gedrückt, so geht das Programm automatisch zum Einlesen des aktuellen Joystickwertes über.

4/4.3.1.1.8

Zeichnen

Das erste, was man mit dem vorliegenden Programm machen möchte, ist freihändiges Zeichnen mit dem Joystick. Dies wird durch folgendes Unterprogramm ermöglicht:

```
10000 REM ------------------------------
10010 REM ---    UP: ZEICHNEN         ---
10020 REM ------------------------------
10030 :
10040 PRINT"▚QQQQQQQQQQQQQQQQQQQQQQQQ▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮";
10050 PRINT"____________________▚ZEICHNEN■";
10060 ZU$="Z"
10070 :
10080 RETURN
```

Zunächst wird dem Anwender in der unteren rechten Bildschirmecke deutlich gemacht, daß der Zeichen-Modus eingeschaltet ist. Außerdem wird in der Variablen ZU$ noch ein Merker gesetzt, der in Zeile 3190 die Ausgabe eines Punktes veranlaßt.

Sicherlich hätte man das vorliegende Unterprogramm auslassen können, und sofort durch Drücken des Feuerknopfes ein Zeichnen realisieren können. Um Ihnen jedoch später die Möglichkeit zu geben, dem Joystick (inklusive Feuerknopf) auch andere Tätigkeiten zuzuordnen, haben wir die vorliegende Form gewählt. Bei einer Änderung der Tätigkeit des Joystickes ist die Zustandsvariable ZU$ auf jeden Fall umzusetzen.

Denkbar für eine solche Anwendung wäre das Abtasten eines Teiles des Grafikbildes mit dem Joystick und Übernahme in eine Zeichenreihe unter Zuhilfenahme des

des SSHAPE-Befehles. Aber auch andere Anwendungen sind denkbar, und Ihrer Phantasie sind dabei keine Grenzen gesetzt.

4/4.3.1.1.9

PAINT — mit Farbe füllen

Der nächste Menüpunkt (P) gibt Ihnen die Möglichkeit, umrandete Gebiete mit einer Farbe auszufüllen. Um Komplikationen zu vermeiden, sollten Sie — soweit möglich — das Gebiet mit einer einzigen Farbe zeichnen. Besteht die Umrahmung zum Teil aus der beim PAINT-Befehl gewählten Farbe und zum Teil aus einer anderen — vom Hintergrund verschiedenen — Farbe, so liegt keine gültige Eingrenzung für den PAINT-Befehl vor. Die einzige Ausnahme ist gegeben, wenn Sie die Umgebung aus zwei Farben zeichnen, die beide nicht die Hintergrundfarbe oder die beim PAINT-Befehl gewählte Farbe darstellen. In diesem Fall ist der Modus 1 zu wählen.

Nach dem Zeichnen eines Gebietes sollten Sie auf jeden Fall den Grafik-Cursor innerhalb des Gebietes bewegen, da sonst der PAINT-Befehl wirkungslos bleibt.

Um den Menübereich in unserem Programm nicht überzustrapazieren, haben wir eine Vorgehensweise gewählt, bei der Sie die Eingaben blind auf der Tastatur eingeben müssen. Aber hier zuerst das Listing:

```
11000 REM ---------------------------------
11010 REM ---    UP: AUSFUELLEN        ---
11020 REM ---------------------------------
11030 :
11040 GETKEY C$
11050 C=VAL(C$)
11060 IF C>3 THEN 11040
11070 :
11080 GETKEY D$
11090 D=VAL(D$)
11100 IF D>1 THEN 11080
11110 :
11120 PAINT C,AX,AY,D
11130 :
11140 RETURN
```

Da hier unbedingt eine Eingabe vorliegen soll, kann man im Gegensatz zur Tastaturabfrage den GETKEY-Befehl verwenden. Zunächst wird in die Variable C$ ein

Wert eingelesen, anschließend in eine Zahl umgewandelt (Zeile 11050) und in Zeile 11060 einer Plausibilitätsprüfung unterzogen, damit nach Verlassen dieser Zeile kein ungültiger Wert vorliegt.

Gleiches geschieht ab Zeile 11080 mit der Variablen D$/D, die nur die Werte 0 oder 1 annehmen darf. Der Tastendruck auf irgendeine Buchstabentaste führt übrigens zum Ergebnis 0, was korrekt ist.

In Zeile 11120 wird schließlich der PAINT-Befehl ausgeführt, wobei die zuerst gewählte Zahl die Farbquelle darstellt und die letzte Zahl den Modus. Obwohl die Position des Grafik-Cursors beim PAINT-Befehl voreingestellt ist, habenwir hier aus Dokumentationsgründen noch die entsprechenden Werte AX und AY angegeben.

4/4.3.1.1.10

Farbe ändern

Sollten einem Anwender die voreingestellten Farben (siehe Zeilen 1060 bis 1110) nicht genehm sein, so hat er jederzeit während des Programmablaufes, und damit auch während des Zeichnens, die Möglichkeit, die voreingestellten Farben zu ändern. Nach dem Drücken eines „F“ im Hauptmenü erscheint unten rechts am Bildschirm die Abfrage „NR. ‚FARBE‘ die entsprechend zu beantworten ist.

```
12000 REM ------------------------------------
12010 REM ---    UP: FARBE AENDERN    ---
12020 REM ------------------------------------
12030 :
12040 PRINT"[illegible]";
12050 :
12060 INPUT"NR.,FARBE          [illegible]";NR,FA
12070 IF NR<0 OR NR>3 THEN 12060
12080 IF FA<1 OR FA>16 THEN 12060
12090 :
12100 COLOR NR,FA
12110 :
12120 RETURN
```

In den Zeilen 12070 und 12080 wird für die beiden Variablen noch eine Plausibilitätsprüfung durchgeführt, die unbedingt notwendig ist, da bei einer falschen Eingabe das Programm abbricht, und die Grafik somit verloren ist. Ist alles korrekt,

so wird mit dem COLOR-Befehl in Zeile 12100 die Farbänderung durchgeführt. Anders als im 64er Modus werden jedoch nicht alle Vorkommen der gewählten Farbe umgefärbt (bei Farbänderung von blau auf schwarz wird also nicht alles bisher blau dargestellte schwarz). Vielmehr können Sie bei geschicktem Umgang alle 16 Farben auf dem Bildschirm gleichzeitig darstellen. Dabei müssen Sie jedoch aufpassen, daß innerhalb einer 8x8 Punktmatrix in hochauflösender Grafik und einer 4x8 Punktmatrix im Mehrfarbenmodus nur vier verschiedene Farben möglich sind (einschließlich Hintergrundfarbe). Warum dies so ist, kann an dieser Stelle wegen der komplizierten Grafik-Struktur nicht näher beleuchtet werden.

4/4.3.1.1.11

Stiftfarbe ändern

Da beim Zeichnen jeweils nur mit einer Farbe gearbeitet werden kann, gibt Ihnen folgendes Unterprogramm die Möglichkeit, zwischen den vier verschiedenen Farbstiften auszuwählen:

```
13000 REM ------------------------------
13010 REM ---    UP: STIFTFARBE      ---
13020 REM ------------------------------
13030 :
13040 GETKEY F$
13050 F=VAL(F$)
13060 IF F>3 THEN 13040
13070 :
13080 RETURN
```

Auch hier wird die Stiftnummer (0-3) wieder mit dem GETKEY-Befehl von der Tastatur eingelesen und auf seine Richtigkeit überprüft. Da wir zur Umwandlung von F$ sogleich die Variable F heranziehen, braucht keine weitere Aktivität durchgeführt zu werden.

Durch dieses Unterprogramm ist es auch möglich, bereits Gezeichnetes zu löschen, indem die Stiftnummer „0“ vorgegeben wird. Dabei ist es sogar möglich — wie wir später noch sehen werden — die Hälfte eines breiten Pinselstriches zu löschen, während der Rest stehen bleibt.

4/4.3.1.1.12

WIDTH — Pinselstärke ändern

Zum Ändern der Pinselstärke reicht ein kleines Programm, da es hier nur zwei Möglichkeiten gibt. Statt des Anwählens über einen Tastendruck oder die Erfassung mittels INPUT-Befehl, wollen wir die Pinselstärke bei Drücken der Taste „W" alternieren lassen. Es soll also jeweils ein Umschalten auf die andere Pinselstärke erfolgen.

```
14000 REM ------------------------------
14010 REM ---    UP: PINSELSTAERKE    ---
14020 REM ------------------------------
14030 :
14040 IF W=1 THEN W=2 : GOTO 14070
14050 IF W=2 THEN W=1
14060 :
14070 WIDTH W
14080 :
14090 RETURN
```

4/4.3.1.1.13

Position merken

Bei manchen Ausgaben ist es wichtig, daß man sich zuvor eine zweite Position gemerkt hat. Dies werden wir im weiteren beim Zeichnen eines Rechteckes und beim Übernehmen eines Grafikteilstückes noch verwenden. Generell kann man nach der Verwendung des gemerkten Punktes zwar diesen löschen, in einigen Fällen ist es jedoch auch sinnvoll, sich die gemerkte Position auch weiterhin in einer Variablen vorrätig zu halten. Wer will, kann sich auch ein kleines Unterprogramm schreiben, das nach Aufruf im Hauptmenü die gemerkte Position löscht und Sprite 2, das wir im Folgenden verwenden wollen, ausschaltet.

```
15000 REM ----------------------------------
15010 REM ---    UP: MERKEN          ---
15020 REM ----------------------------------
15030 :
15040 PRINT"[Cursor-Steuerzeichen]";
15050 PRINT"GEMERKT: X=";AX;";Y=";AY;"_|";
15060 :
15070 MX=AX
15080 MY=AY
15090 :
15100 SPRITE 2,1,1
15110 MOVSPR 2,AX*2+13,AY+40
15120 :
15130 RETURN
```

In manchen Fällen ist es sinnvoll, neben einer Markierung im Grafik-Bildschirm auch die Position in absoluten Zahlen anzugeben. Dies wird durch die beiden Zeilen 15040 und 15050 bewerkstelligt, wobei in der ersten der beiden Zeilen der Cursor entsprechend positioniert wird und in der nächsten Zeile die Ausgaben der Daten erfolgt. Da die Ausgabe in der unteren Bildschirmzeile erfolgt, wurde darauf geachtet, daß der Rest des Randes erhalten bleibt. Wenn Sie vom Unterprogramm ‚Merken' zum Unterprogramm ‚ZEICHNEN' übergehen, so wird dies in Zeile 10050 ebenfalls berücksichtigt.

Als mnemotechnische Bezeichnung für die gemerkten X- und Y-Koordinate bieten sich die Variablen MX und MY geradezu an. An dieser Stelle wird auch unser zweites definiertes Sprite verwendet, indem es in Zeile 15100 zunächst eingeschaltet und in Zeile 15110 an der entsprechenden Position im Grafik-Bildschirm seinen Platz erhält. Durch die Zeichnung innerhalb des Sprites ergeben sich gegenüber den vorher genannten Umrechnungsarten Spritepositionen/Grafikposition einige Änderungen, da sich nicht die linke obere Ecke des Sprites sondern der Mittelpunkt unseres ‚X' über der gemerkten Position befinden soll. Die Zeichnung unsers Sprites wurde so gewählt, daß sich der Mittelpunkt des Sprites über der zu merkenden Position befindet, Sie also auch Sprites nach Ihren Wünschen (eingekreistes X, nur Kreis) definieren können, um die gleiche Wirkung zu erzielen.

4/4.3.1.1.14

BOX — Rechtecke und Blöcke

Neben dem DRAW-Befehl wollen wir an dieser Stelle auch noch einen weiteren Befehl aus dem Bereich der grafischen Figuren vorstellen. Die Möglichkeiten der Kreisdarstellung überlassen wir einer Ergänzung des Programms durch Sie.

```
16000 REM ------------------------------------
16010 REM ---     UP: BOX                  ---
16020 REM ------------------------------------
16030 :
16040 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒████████████████████████";
16050 :
16060 INPUT"WINKEL,MAL             ██████████";WI,M
16070 IF M<>0 AND M<>1 THEN 16060
16080 :
16090 BOX F,MX,MY,AX,AY,WI,M
16100 :
16110 RETURN
```

Da der Winkel, um den eventuell das Rechteck oder der Block gedreht werden soll, nicht mit einem einzigen Tastendruck zu erfassen ist, benutzen wir wieder die unterste Zeile unseres Menübereiches zur Erfassung des Winkels und der Farbe, mit der das Rechteck eventuell ausgefüllt werden soll. Zeile 16070 beinhaltet eine Plausibilitätsprüfung, die wiederum einen Programmabbruch verhindert und in Zeile 16090 wird das Rechteck (ausgefüllt oder nicht) gezeichnet. Dabei benutzen wir die im vorigen Unterprogramm gemerkten Koordinaten MX und MY. Achten Sie darauf, daß Sie vor dem Zeichnen eines Rechteckes/BLockes an der oberen linken Ecke die Koordinaten durch „Merken“ übernehmen, da sonst — z.B. nach Programmstart — die Koordinaten 0,0 angenommen werden, was die obere linke Ecke des Grafik-Bildschirmes bedeutet.

Hier nun ein kleines Beispiel im Vorgriff auf unser Kapitel über grafische Figuren. Bewegen Sie den Grafik-Cursor zunächst an die Position 50,40, was Ihnen durch die Positionsangabe am rechten oberen Rand des Textbildschirmes erleichtert wird. Drücken Sie dann die Taste „M“, es erscheint Sprite 2 an dieser gemerkten Position. Führen Sie nun den Cursor zu der Position 100,80 und drücken Sie „B“ und „0,0“. Es erscheint nun ein Rechteck.

Wenn Sie jetzt nochmals „B“ drücken und eingeben „20“, so erscheint das Rechteck leicht gedreht. Dabei braucht der Wert für den Malstift nicht nochmals angegeben zu werden, da er sich nicht ändert. Wenn Sie im Folgenden das Rechteck durch fortlaufendes Drücken von „B“ und einem Winkel mit folgendem Komma eingeben,

wobei der Winkel immer um 20 Grad ansteigen sollte, so erhalten Sie ein Muster, das einem gehäkelten Deckchen nicht unähnlich ist.

4/4.3.1.1.15

SSHAPE — Grafikdaten an Zeichenreihe übergeben

Auch die Befehlskombination SSHAPE/GSHAPE wollen wir innerhalb dieses Programmes vorstellen. Durch diese beiden Befehle erhält man vielfältige Möglichkeiten, vorhandene Grafiken zu ändern, Teilgrafiken zu speichern und in einem anderen Programm zu laden, aber auch die Möglichkeit des invertierens eines Teilbereiches — an der gleichen Stelle — ist durch diese beiden Befehle möglich.

Zunächst muß jedoch ein Teilbereich der Grafik in eine Zeichenkette verwandelt werden, was mit dem SSHAPE-Befehl geschieht. Da Zeichenreihen jedoch nicht mehr als 255 Zeichen enthalten können, ist die Übernahme eines Grafikbereiches sehr stark eingeschränkt. Die beiden Formeln im Handbuch sagen letztendlich nur aus, daß für die Übernahme einer Zeichnung aus dem normalen hochauflösenden Grafik-Modus acht Bildschirmpunkte in einem Zeichen (Byte) untergebracht werden können. Zusätzlich wird noch einige Randinformation übernommen, wie die Summanden .99 und 4 zeigen.

Beim Mehrfarbenmodus können wegen der Farbdarstellung in je zwei Bit nur die Hälfte der ansprechbaren Bildpunkte übernommen werden. In beiden Fällen wird also ein gleich großer Ausschnitt aus dem Grafik-Bildschirm in eine Zeichenreihe übertragen.

```
17000 REM ----------------------------
17010 REM ---    UP: UEBERNEHME      ---
17020 REM ----------------------------
17030 :
17040 IF INT((ABS(AX-MX)+1)/4+.99)*(ABS(AY-MY)+1)+4 > 255 THEN RETURN
17050 :
17060 SSHAPE SS$,MX,MY
17070 :
17080 RETURN
```

Unsere Formel in Zeile 17040 ist nichts anderes, als die Überprüfung — im Bezug auf den Mehrfarbenmodus —, ob die zu übernehmende Zeichenreihe nicht länger

als 255 wird, was einen Programmabbruch verursachen würde. Liegt ein solcher Fall vor, wird das Unterprogramm verlassen, ohne daß eine Transaktion durchgeführt wird.

Im anderen Fall werden die Daten in die Zeichenreihe SS$ übernommen, wobei die im Handbuch angegebene Positionsangabe x2, y2 durch die Position des Grafik-Cursors ersetzt wird und der Anfang der Zeichenreihe durch die gemerkte Position MX, MY festgelegt ist.

4/4.3.1.1.16

GSHAPE — Zeichenreihe in Grafik einfügen

Der umgekehrte Fall zum Befehl SSHAPE ist mit dem GSHAPE-Befehl gegeben. Im folgenden Programm benutzen wird die schon einige Male aufgeführte Vorgehensweise, indem wir den im GSHAPE-Befehl zu verwendenden Modus wieder über A$/A einlegen und auf Plausibilität prüfen.

```
18000 REM ------------------------------
18010 REM ---    UP: KOPIERE         ---
18020 REM ------------------------------
18030 :
18040 GETKEY A$
18050 A=VAL(A$)
18060 IF A>4 GOTO 18040
18070 :
18080 GSHAPE SS$,AX,AY,A
18090 :
18100 RETURN
```

In Zeile 18080 wird der Befehl ausgeführt, wobei wir aus Dokumentationsgründen die Position des Grafik-Cursors in den Variablen AX und AY innerhalb des Befehles mitangeben.

Bei der Bestimmung des Modus sollten Sie die Angaben im Handbuch sorgfältig durchlesen. Besonders der Modus 4 (Invertieren) ist sehr interessant, wie Sie nach einigem Ausprobieren sicherlich selbst feststellen werden. Das Invertieren geschieht allerdings nicht aufgrund der Farbzuordnung, sondern im Bitmuster. Wenn Sie ein übernommenes Bild durch die Eingaben „K“ und „O“ an eine andere Position gebracht haben, so können Sie mit „K“ und „1“ dieses Bild invertieren, wobei die — wenn die Farben nicht geändert wurden — blaue Zeichnung auf gelbem Grund durch eine grüne Zeichnung auf braunem Grund ersetzt wird.

Durch „K“ und „O“ erhalten Sie den Ursprungszustand zurück. Wenn Sie nach dem invertierten Bild (durch K/1) anschließend sofort „K“ und „2“ eingeben, so erhalten Sie lediglich ein braunes Rechteck (ausgefüllt). Aus dieser Darstellung erhalten Sie durch „K“ und „4“ wieder das originäre Bild und mit „K“ und „5“ wird der entsprechende Bereich am Bildschirm gelöscht.

Durch die Möglichkeit des Invertierens können Sie mit Hilfe der durch SSHAPE gespeicherten Zeichenreihen auch Sprites konstruieren, die Sie durch fortlaufendes Invertieren auch auf dem Bildschirm bewegen können. Dies ist z.B. dann ein Vorteil, wenn die Größe der Sprites für die gewünschte Darstellung nicht ausreicht, oder mehr als acht Sprites für eine Anwendung nötig sind. Es empfiehlt sich, mit den Befehlen SSHAPE und GSHAPE auch außerhalb dieses Programmes ausgiebig zu experimentieren.

4/4.3.1.1.17

Bedienungsanleitung

Für diejenigen, die nur das Gesamtlisting übernehmen wollen, ohne die einzelnen Unterkapitel 4/4.3.1.1.1 bis 4/4.3.1.1.16 durchzuarbeiten, aber auch für die anderen zum Nachschlagen hier noch einmal die Bedienungsanleitung.

Zeichnen

Nachdem Sie das Programm gestartet haben, erscheint in der Mitte des Bildschirms der Zeiger auf den Grafik-Cursor und im linken oberen Teil des Textbildschirmes die dazugehörigen Koordinaten in inverser Schrift. Wenn Sie nun die Taste „Z“ drücken und den Joystick mit gedrücktem Feuerknopf bewegen, so wird die Bahn des Grafik-Cursors nachgezeichnet. Sobald Sie den Feuerknopf loslassen, können Sie den Grafik-Cursor bewegen, ohne daß seine Bewegung nachvollzogen wird.

Pinselstärke ändern

Durch Drücken auf die Taste „W“ können Sie die Pinselstärke alternierend zwischen einfacher und doppelter Stärke wählen. Haben Sie vor dem Druck auf „W“ einfache Pinselstärke, so zeichnen Sie anschließend mit der doppelten Pinselstärke und umgekehrt.

Stiftfarbe ändern

Die Stiftfarbe können Sie durch Drücken der Taste „S“ ändern, wobei Sie anschließend eine der Zifferntasten 0-3 drücken müssen. Andere Eingaben werden ignoriert. Daß eine weitere Einfabe vom Rechner erwartet wird, können sie immer daran erkennen, daß die Positionsangabe nicht flackert. Das Flackern rührt von der Ausgabe des aktuellen Standes her, die unterbrochen wird, wenn der Rechner eine Eingabe erwartet.

Auch die Stiftfarbe 0 ist zugelassen, womit Sie in Ihrer Zeichnung auch „Radieren“ können. Jede Buchstabentaste bzw. jedes Sonderzeichen wird als Stiftfarbe 0 interpretiert.

Farbe ändern

Nach Druck auf die Taste „F“ erscheint unten rechts am Bildschirmrand die Abfrage „NR., FARBE?“ Geben Sie bitte zunächst eine der Farbstiftnummern zwischen 0 bis 3 ein und anschließend eine der Farbennummern 1 bis 56. Ungültige Eingaben werden ignoriert und führen zur erneuten Abfrage.

Gebiet ausmalen

Sofern Sie ein Gebiet gezeichnet haben und dies ausmalen wollen, so drücken Sie die Taste „P“. Der Rechner erwartet danach die Nummer des Stiftes, mit dem der Bereich ausgefüllt werden soll. Ist auch diese Nummer eingegeben, so muß eine der Ziffern „0“ oder „1“ eingegeben werden, was dem Modus im PAINT-Befehl entspricht. Um Komplikationen zu vermeiden, sollten die Gebiete mit der gleichen Farbe umrandet sein.

Rechtecke und Blöcke zeichnen

Wenn Sie ein Rechteck bzw. einen Block ausgeben wollen, so bewegen Sie den Grafik-Cursor auf die linke obere Ecke der gewünschten Ausgabe und drücken Sie anschließend die Taste „M“. Es erscheint ein „X“, mit dem die gemerkte Stelle gekennzeichnet wird. Außerdem wird am unteren Bildschirmrand die Position in Koordinaten ausgegeben.

Bewegen Sie dann den Cursor auf die gewünschte untere rechte Ecke und drücken Sie die Taste „B“. Danach erfragt der Rechner den Drehwinkel und eine Angabe, ob das Rechteck ausgefüllt werden soll (1) oder nicht (0).

Die gemerkte Position bleibt auch nach Ausführung des BOX-Befehls erhalten, so daß Sie damit weiter arbeiten können.

Bildschirmbereiche kopieren

Unter Verwendung der Befehle SSHAPE und GSHAPE wird ein Kopieren von Bildschirmbereichen ermöglicht, indem eine Zeichenreihe zu Hilfe genommen wird.

Beachten Sie bitte die zulässige Größe der Zeichenreihe von 255 Zeichen, was im hochauflösenden Grafik-Modus ca. 2000 Bildschirmpunkten entspricht und im Mehrfarbenmodus ca. 1000 Bildschirmpunkten.

Ebenso wie beim Zeichnen von Rechtecken/Blöcken ist auch hier wiederum zunächst die linke obere Ecke des zu übernehmenden Bildschirmbereiches durch Druck auf die Taste „M“ zu merken. Anschließend muß der Cursor auf die gewünschte rechte untere Ecke gebracht werden und es ist die Taste „U“ zu drücken.

Dann können Sie den Grafik-Cursor in die linke obere Ecke der gewünschten Ausgabe bewegen, um die Taste „K“ fürs Kopieren zu drücken. Beim Überschreiben des Bildschirmbereiches wird der GSHAPE-Befehl benutzt, so daß nach dem „K“ noch eine der Tasten 0 bis 4 zu drücken ist, um dem Rechner auch den gewünschten Modus mitzuteilen.

Soweit die vorgegebenen Features des Programmes. Ergänzungen durch Sie sind durch den strukturierten Programmaufbau leicht möglich.

4/4.3.1.1.18

Gesamtlisting

```
1000 REM -----------------------------
1010 REM --- ZEICHENPROGRAMM       ---
1020 REM -----------------------------
1030 :
1040 GRAPHIC 4,1,20
1050 :
1060 COLOR 0,8
1070 COLOR 1,7
1080 COLOR 2,6
1090 COLOR 3,9
1100 COLOR 4,8
1110 COLOR 5,1
1120 :
1130 DRAW 1,0,0 TO 0,159 TO 159,159 TO 159,0 TO 0,0
```

Listing 4/4.3.1.1 Zeichnen (1)

```
1150 PRINT"┌                                          ┐";
1160 PRINT"|                                          |";
1170 PRINT"|                                          |";
1180 PRINT"|                                          |";
1190 PRINT"└                                          ┘";
1200 :
2000 REM -------------------------------
2010 REM ---   MENUE  UND ZEIGER    ---
2020 REM -------------------------------
2030 :
2040 PRINT"SQQQQQQQQQQQQQQQQQQQQ";
2050 PRINT"Z-ZEICHNEN  M-MERKEN   S-STIFT         |";
2060 PRINT"B-BOX       P-AUSFUEL. W-PINSEL        |";
2070 PRINT"K-KOPIEREN  F-FARBE    U-UEBERNEHMEN   |";
2080 :
2090 REM  BLOAD"ZEICHSPRITE"
2100 :
2110 SPRITE 1,1,1,0,0,0,0
2120 MOVSPR 1,184,110
2130 :
2140 LOCATE 80,60
2150 :
2160 AX=80
2170 AY=60
2180 :
2190 F=1
2200 :
2210 W=1
2220 :
3000 REM -------------------------------
3010 REM ---   JOYSTICKABFRAGE      ---
3020 REM -------------------------------
3030 :
3040 JO=JOY(2)
3050 :
3060 IF JO>127 THEN TA=1 : JO=JO-128
3070 :
3080 IF JO=1 THEN AY=AY-1 : GOTO 3170
3090 IF JO=2 THEN AX=AX+1 : AY=AY-1 : GOTO 3170
3100 IF JO=3 THEN AX=AX+1 : GOTO 3170
3110 IF JO=4 THEN AX=AX+1 : AY=AY+1 : GOTO 3170
3120 IF JO=5 THEN AY=AY+1 : GOTO 3170
3130 IF JO=6 THEN AX=AX-1 : AY=AY+1 : GOTO 3170
3140 IF JO=7 THEN AX=AX-1 : GOTO 3170
3150 IF JO=8 THEN AX=AX-1 : AY=AY-1 : GOTO 3170
3160 :
3170 MOVSPR 1,AX*2+24,AY+50
3180 :
3190 IF TA THEN TA=0 : IF ZU$="Z" THEN DRAW F,AX,AY
3200 :
3210 IF AX<0   THEN AX=0
3220 IF AX>159 THEN AX=159
3230 IF AY<0   THEN AY=0
3240 IF AY>199 THEN AY=199
3250 :
3260 PRINT"SQQQQQQQQQQQQQQQQQQQQQQQ";
3270 PRINT"3         X:";AX
3280 PRINT"SQQQQQQQQQQQQQQQQQQQQQQQQ";
3290 PRINT"3         Y:";AY
3300 :
```

Listing 4/4.3.1.1 Zeichnen (2)

```
4000 REM -----------------------------
4010 REM ---    TASTATURABFRAGE      ---
4020 REM -----------------------------
4030 :
4040 GET A$
4050 IF A$<>"" THEN B$=A$ : A$="": ELSE GOTO 3040
4060 :
4070 IF B$="Z" THEN GOSUB 10000 : GOTO 3040
4080 IF B$="P" THEN GOSUB 11000 : GOTO 3040
4090 IF B$="F" THEN GOSUB 12000 : GOTO 3040
4100 IF B$="S" THEN GOSUB 13000 : GOTO 3040
4110 IF B$="W" THEN GOSUB 14000 : GOTO 3040
4120 IF B$="M" THEN GOSUB 15000 : GOTO 3040
4130 IF B$="B" THEN GOSUB 16000 : GOTO 3040
4140 IF B$="U" THEN GOSUB 17000 : GOTO 3040
4150 IF B$="K" THEN GOSUB 18000 : GOTO 3040
4160 IF B$="L" THEN RUN
4170 :
4180 B$=""
4190 :
4200 GOTO 3040
4210 :
10000 REM -----------------------------
10010 REM ---    UP: ZEICHNEN         ---
10020 REM -----------------------------
10030 :
10040 PRINT"■QQQQQQQQQQQQQQQQQQQQQQQQ■■■■■■■■■■■■■■■■■■■";
10050 PRINT"________________________■ZEICHNEN■";
10060 ZU$="Z"
10070 :
10080 RETURN
10090 :
11000 REM -----------------------------
11010 REM ---    UP: AUSFUELLEN       ---
11020 REM -----------------------------
11030 :
11040 GETKEY C$
11050 C=VAL(C$)
11060 IF C>3 THEN 11040
11070 :
11080 GETKEY D$
11090 D=VAL(D$)
11100 IF D>1 THEN 11080
11110 :
11120 PAINT C,AX,AY,D
11130 :
11140 RETURN
11150 :
12000 REM -----------------------------
12010 REM ---    UP: FARBE AENDERN    ---
12020 REM -----------------------------
12030 :
12040 PRINT"■QQQQQQQQQQQQQQQQQQQQQQQQ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■";
12050 :
12060 INPUT"NR.,FARBE          ■■■■■■■■■■";NR,FA
12070 IF NR<0 OR NR>3 THEN 12060
```

Listing 4/4.3.1.1 Zeichnen (3)

```
12080 IF FA<1 OR FA>16 THEN 12060
12090 :
12100 COLOR NR,FA
12110 :
12120 RETURN
12130 :
13000 REM ----------------------------------------
13010 REM ---    UP: STIFTFARBE              ---
13020 REM ----------------------------------------
13030 :
13040 GETKEY F$
13050 F=VAL(F$)
13060 IF F>3 THEN 13040
13070 :
13080 RETURN
13090 :
14000 REM ----------------------------------------
14010 REM ---    UP: PINSELSTAERKE           ---
14020 REM ----------------------------------------
14030 :
14040 IF W=1 THEN W=2 : GOTO 14070
14050 IF W=2 THEN W=1
14060 :
14070 WIDTH W
14080 :
14090 RETURN
14100 :
15000 REM ----------------------------------------
15010 REM ---    UP: MERKEN                  ---
15020 REM ----------------------------------------
15030 :
15040 PRINT"[illegible]";
15050 PRINT"GEMERKT: X=";AX;";Y=";AY;"_";
15060 :
15070 MX=AX
15080 MY=AY
15090 :
15100 SPRITE 2,1,1
15110 MOVSPR 2,AX*2+13,AY+40
15120 :
15130 RETURN
15140 :
16000 REM ----------------------------------------
16010 REM ---    UP: BOX                     ---
16020 REM ----------------------------------------
16030 :
16040 PRINT"[illegible]";
16050 :
16060 INPUT"WINKEL,MAL          [illegible]";WI,M
16070 IF M<>0 AND M<>1 THEN 16060
16080 :
16090 BOX F,MX,MY,AX,AY,WI,M
16100 :
16110 RETURN
16120 :
17000 REM ----------------------------------------
17010 REM ---    UP: UEBERNEHME              ---
17020 REM ----------------------------------------
17030 :
```

Listing 4/4.3.1.1 Zeichnen (4)

```
17040 IF INT((ABS(AX-MX)+1)/4+.99)*(ABS(AY-MY)+1)+4 > 255 THEN RETURN
17050 :
17060 SSHAPE SS$,MX,MY
17070 :
17080 RETURN
17090 :
18000 REM ---------------------------------
18010 REM ---    UP: KOPIERE          ---
18020 REM ---------------------------------
18030 :
18040 GETKEY A$
18050 A=VAL(A$)
18060 IF A>4 GOTO 18040
18070 :
18080 GSHAPE SS$,AX,AY,A
18090 :
18100 RETURN
```

Listing 4/4.3.1.1 Zeichnen (5)

4/4.3.1.1.19

Variablenübersicht

A	Hilfsvariable für Modus bei GSHAPE (0-4)
A$	Eingelesenes Zeichen für Tastaturabfrage
AX	Aktuelle Grafik-Cursorposition in X-Richtung (0-159)
AY	Aktuelle Grafik-Cursorposition in Y-Richtung (0-199)
B$	Gültiges Zeichen in Tastaturabfrage
C/C$	Hilfsvariablen zum Einlesen der Stiftfarbe beim Ausmalen (0-3)
D/D$	Hilfsvariablen zum Einlesen des Modus beim Ausmalen (0,1)
F	Aktuelle Farbquelle/Zeichenstiftnummer (0-3)
F$	Hilfsvariable zum Einlesen der Stiftnummer bei Stiftauswahl (0-3)
FA	Hilfsvariable zum Einlesen der Farbstiftnummer bei Farbänderung (1-16)

JO	Eingelesener Joystickwert (0-8, 128-136)
M	Merker für Box ausmalen (0,1)
MX	Gemerkte X-Koordinate (0-159)
MY	Gemerkte Y-Koordinate (0-199)
NR	Hilfsvariable zum Einlesen der Farbstiftnummer bei Farbänderung (0-3)
SS$	Übernommene Grafikdaten mit SSHAPE
TA	Merker für Feuertaste (0,1)
W	Aktuelle Pinselstärke (0,1)
WI	Winkel beim Zeichnen eines Rechteckes/Blocks (0-360)
ZU$	„Zustand" ("", Z)

4/4.3.2

Sprites

Eingegliedert in den Bereich der hochauflösenden Grafik, jedoch als eigene Grafikform betrachtet werden müssen die Sprites. Sprites sind besonders für Spiele interessant, da freidefinierbare Figuren oder andere Zeichnungen sehr schnell über den Bildschirm bewegt werden können. Aber auch in anderen Anwendungen bieten Sprites durchaus Vorteile, wie wir an anderer Stelle noch zeigen werden.

Das Basic 7.0 des C 128 stellt so viele Sprite-Befehle zur Verfügung, daß keine Wünsche offen bleiben. Manche Befehle (z.B. SPRDEF) beinhalten ein ganzes Programm, wie man sich es z.B. beim C 64 selbst schreiben mußte.

4/4.3.2.1

SPRDEF — Der Entwurf

Nach Aufruf des Befehls SPRDEF (nur in Verbindung mit 40-Zeichen-Monitor möglich) steht Ihnen ein Sprite-Editor zur Verfügung, mit dem Sie Sprites beliebig nach Ihren Wünschen entwerfen können, wobei für hinreichendes Handwerkszeug gesorgt ist. Insbesondere die Möglichkeit des Speicherns mittels BSAVE, wie im Handbuch angegeben, ist hier hervorzuheben, da auf diese Art und Weise Sprites getrennt von Programmen auf Diskette gespeichert werden können. Diese Unabhängigkeit vom Programm erlaubt es, Sprites auch in mehreren Programmen zu verwenden, diese jedoch nur einmal zu definieren. Außerdem entfällt die lästige POKErei und Speicherplatz im Hauptspeicher wird durch die entfallenden DATA-Zeilen ebenfalls gespart (obwohl das beim C 128 nicht so sehr ins Gewicht fällt).

Eine kleine Schönheitskorrektur würde die Arbeit jedoch noch ein wenig verbessern, nämlich dann, wenn die Sprites in Originalgröße sowohl nicht vergrößert, als

auch vergrößert in beide Richtungen sowie vergrößert in die eine der Richtungen dargestellt würden. Hier könnte bei gleichzeitigem Betrachten der vier Vergrößerungsmöglichkeiten schon die Entscheidung für die spätere Verwendung fallen.

Die Verwendung des Sprite-Editors ist im Handbuch ausreichend beschrieben, so daß wir hier in Bild 4/4.3.2-1 eine Idee für ein Sprite vorgeben wollen, wie es in den weiteren Beispielen zu verwenden wäre.

			2								
			2	2							
				2	2			2	2		
3					2	2	2			2	
4	3	2	2	2	2	2	2	2	2	2	2
3	4	2	2	2	2	2	2	2	2	2	2
4	3	2	2	2	2	2	2	2	2	2	2
3					2						
				2	2						
			2	2							
			2								

Bild 4/4.3.2 - 1

In dem Bild wurden jeweils die für den Hintergrund verschiedenen Farbnummern eingetragen, die mit dem beim Sprite-Editor zu drückenden Tasten identisch sind. Nicht eingefärbte Felder erhalten die Farbe 1, entsprechend dem Hintergrund.

Dieses Sprite soll uns im Verlauf der weiteren Befehlserweiterungen begleiten. Das erstellte Sprite sollte deshalb mit dem Befehl BSAVE „TESTSPRITES“ ,B0,P3584 TO P4095 gespeichert werden.

4/4.3.2.2

SPRSAV — Sprites an Zeichenreihe übergeben
Sprites kopieren

Sicherlich würde ein einziges Sprite ausreichen, um die anderen Befehle innerhalb dieses Buches zu erklären. Ein Flugzeuggeschwader ist jedoch ein schönerer Anblick als ein einzelnes Flugzeug und wir werden später auch noch mehrere Sprites benötigen. Es wäre allerdings etwas mühselig, jedes Sprite mit dem gleichen Muster zu versehen, so daß hier nach Abhilfe gesucht werden muß.

Die Abhilfe liegt vor in Form des SPRSAV-Befehls mit dem ein Sprite an eine Zeichenkette übergeben werden kann, und durch Vertauschen der Parameter ebenso eine Zeichenkette einer Spritenummer zugewiesen werden kann. Um nun aus unserem einzelnen Flugzeug ein Geschwader von acht Flugzeugen zu machen, muß zunächst das Datenfeld der Sprites an eine Zeichenreihe übergeben werden, was wir mit

```
SPRSAV 1,O$
```

durchführen. Das 'O' steht hier für Original, da wir später mit den Spritedaten auch noch etwas anderes durchführen werden. Durch Vertauschen der Parameter, also:

```
SPRSAV O$,2
SPRSAV O$,3
SPRSAV O$,4
SPRSAV O$,5
SPRSAV O$,6
SPRSAV O$,7
SPRSAV O$,8
```

haben wir unser definiertes Sprite allen acht Spritenummern zugordnet, wie Sie sich mit Hilfe des SPRDEF-Befehles veranschaulichen können. Umsteigern vom C 64 sei hier geraten, auf die um eine Einheit versetzte Numerierung der Sprites (1-8 statt 0-7) zu achten.

Die so erhaltenen acht Sprites sollten mit dem BSAVE-Befehl — wie oben erwähnt — auf Diskette gesichert werden.

Bevor wir nun die Sprite-Befehle weiter besprechen, sei hier angemerkt, daß die in Zeichenreihen übergebenen Sprites ebenso mit dem GSHAPE-Befehl sichtbar gemacht werden können, wie das folgende kleine Beispiel zeigt:

```
GRAPHIC 3,1
GSHAPE O$,10,10,0
```

Auch die anderen Modi sollten Sie durch Probieren prüfen. Durch Verwendung der Modi 0 und 4 ist es auch möglich, ein Sprite (in diesem Falle nicht mehr im eigentlichen Sinne) am Bildschirm darzustellen und wieder zu löschen und per Basic-Programm läßt sich auf diese Art und Weise auch eine Bewegung erzeugen, wie folgendes kurze Programmstück zeigt (Achten Sie darauf, daß die Variable O$ noch ihren Inhalt hat und starten Sie das kurze Programmstück mit 'GOTO 100'):

```
100 GRAPHIC 3,1
110 FOR I=1 TO 100
120 GSHAPE O$,I,80,0
130 GSHAPE O$,I,80,4
140 NEXT
```

Natürlich ist diese Bewegung nicht sehr elegant und wir werden im weiteren Verlauf des Kapitels über Sprites noch eine bessere Möglichkeit kennenlernen, da u.a. durch die Bewegung auch das ganze Programm gebunden ist und kein zweites Sprite bewegt werden kann, es sei denn, man verschachtelt die Ausgaben ineinander, was jedoch einen noch langsameren Ablauf bedeuten würde.

Auf jeden Fall sollte man sich diesen Trick aber merken, falls mehr als acht Sprites gewünscht werden, oder die Dimension eines Sprites mit 12 x 21 bzw. 24 x 21 Bildpunkten nicht ausreichend ist.

Wenn Sie sich mit

```
?LEN(O$)
```

die Länge der aus dem Sprite erzeugten Zeichenreihe ansehen, so werden Sie feststellen, daß diese 67 beträgt. 64er-Umsteiger werden wissen, daß für die Speicherung eines Sprites 63 Byte bzw. Zeichen benötigt werden. In den vier **letzten** Zeichen sind Angaben über die Länge in X- bzw. Y-Richtung vorhanden, was Sie sich leicht durch folgende Eingabe veranschaulichen können:

```
FOR I= 64 TO 67 : PRINT ASC(MID$(O$I,1)) : NEXT
```

Die darauffolgende Ausgabe lautet:

```
23
0
20
0
```

Wenn man von der Breite (24) und der Höhe (21) jeweils eine Einheit abzieht, so ergeben sich die genannten von Null verschiedenen Zahlen. Es ist anzunehmen, daß die ersten beiden ausgegebenen Byte die Ausdehnung in X-Richtung umschreiben, wobei bei mehr als 255 Bildpunkten Breite das zweite Byte entsprechend einer '1' aufweist. Es ist weiterhin zu vermuten, daß beim SSHAPE-Befehl diese Vorgehensweise ebenfalls gewählt wurde, da die Formeln im Handbuch neben dem Bereich für die Daten noch einen Summanden von vier als letztes aufweisen. Damit ist auch klar, wie die Höhe und Breite eines Grafikausschnittes mit SSHAPE und GSHAPE festgelegt wird, was wir uns noch im Kapitel Tips und Tricks zunutze machen wollen.

4/4.3.2.3

SPRCOLOR — Sprites einfärben

Bevor wir mit unserem Sprite-Kurs fortfahren, sei zunächst auf Kapitel 4/6.2.1.1 hingewiesen, wo das Spiegeln von Sprites erläutert wird. Um unser Beispiel sinnvoller zu gestalten, werden wir im weiteren auch mit gespiegelten Sprites arbeiten.

Umsteiger vom C 64 wird sicherlich bekannt sein, daß zwar Sprites mit maximal vier Farben (inklusive Hintergrund) eingefärbt werden können, davon zwei Farben jedoch allen Sprites gemeinsam sind. Dies sind die Farben, die mit den Tasten '3' und '4' beim Sprite-Editor angewählt werden können, nämlich die zusätzlichen Farben im Mehrfarbenmodus.

Diese beiden Farben können ähnlich dem COLOR-Befehl bei normalen Grafiken mit dem SPRCOLOR-Befehl eingefärbt werden.

Bei unseren bisher erstellten Sprites wollen wir diese beiden Farben mit gelb und hellbraun belegen, was wir mit

```
SPRCOLOR 8,9
```

erledigen.

4/4.3.2.4

SPRITE — Sprite-Parameter festlegen

Einer der wichtigsten Befehle in der Handhabung von Sprites ist der SPRITE-Befehl selbst, der, ebenso wie die anderen Grafikbefehle, die lästige POKErei vom 64er-Modus erspart.

Im folgenden wollen wir unsere Sprites nacheinander einschalten, wobei wir unterschiedliche Parameter für die einzelnen Sprites wählen. Zunächst wollen wir jedoch den Hintergrund den Flugzeugen anpassen und entsprechend mit

```
COLOR,15
```

hellblau einfärben. Den von links nach rechts fliegenden Flugzeugen (Sprite 1 bis 4) wollen wir die Farbe rot, und den anderen Flugzeugen die Farbe dunkelgrau zuweisen. Außerdem sollen die Flugzeuge in unterschiedlichen Größen dargestellt und andere Prioritäten für Vor- und Hintergrund vergeben werden. Eine Vergabe von Attributen könnte demnach wie folgt aussehen:

```
SPRITE 1,1,3,1,0,0,1
SPRITE 2,1,3,1,1,1,1
SPRITE 3,1,3,0,0,0,1
SPRITE 4,1,3,0,1,1,1
SPRITE 5,1,12,1,0,0,1
SPRITE 6,1,12,1,1,1,1
SPRITE 7,1,12,0,0,0,1
SPRITE 8,1,12,0,1,1,1
```

Während Sie die Befehle im Direktmodus eingeben, erscheint nach jeder Eingabe ein neues Sprite auf dem Bildschirm, an einer zuvor nicht näher definierten Position, die von vorhergehenden Tätigkeiten abhängt.

Mit der vorhergehenden Definition haben wir also den jeweiligen Sprites die gewünschte Farbe zugeordnet, und wenn man von diesem Parameter absieht, haben die Sprite 1/5, 2/6, 3/7 und 4/8 jeweils die gleichen Spezifikationen erhalten. Dabei haben die Sprites 1, 2, 5 und 6 jeweils die Priorität 1 zugewiesen bekommen und laufen hinter anderen Objekten am Bildschirm und die Sprites 2, 4, 6 und 8 sind sowohl in X- als auch Y-Richtung ausgedehnt. Allen Sprites gemeinsam ist das Attribut des Mehrfarbenmodus, wie der letzte Parameter anzeigt.

4/4.3.2.5

MOVESPR — Positionieren und Bewegen von Sprites

Nachdem die Sprites zwar am Bildschirm sichtbar sind, aber sicherlich nicht zufällig die gewünschte Position eingenommen haben, wollen wir uns nun zuerst mit der absoluten Positionierung und anschließend mit der Bewegung der Sprites beschäftigen. Hierbei sei darauf hingewiesen, daß die Positionierung der Sprites nichts mit den Grafik-Koordinaten gemein hat und die Sprite-X-Koordinate 0 24 Punkte links neben dem Bildschirmrand liegt, somit Sprites unsichtbar dargestellt werden können. Der obere Rand (Grafik-Y-Koordinate 0) wird durch die Sprite-Koordination 50 angesprochen.

Unser 'rotes Geschwader' wollen wir in der linken unteren Bildschirmhälfte positionieren und unser hellgraues Geschwader entsprechend gegenüber in der rechten unteren Bildschirmhälfte, so daß der Eindruck des aufeinanderzufliegens entsteht. Folgende Koordinaten werden vorgeschlagen:

```
MOVSPR 1,0,2
MOVSPR 2,30,170
MOVSPR 3,100,180
MOVSPR 4,80,150
MOVSPR 5,300,200
MOVSPR 6,340,170
MOVSPR 7,240,180
MOVSPR 8,280,150
```

Dabei ist das erste Sprite unter dem linken Bildschirmrand verborgen und Sprite # 6 ragt nur mit der Nase am rechten Bildschirmrand heraus.

Damit haben wir unsere Geschwader positioniert, bevor jedoch Bewegung ins Spiel kommt, wollen wir noch einen kleinen optischen Effekt beisteuern, der die beiden unterschiedlichen Farben am Ende des Flugzeuges zur Geltung bringt und den Nachbrenner simuliert.

Dazu benutzen wir den schon besprochenen SPRCOLOR-Befehl, indem wir in einer prinzipiellen Endlosschleife mit eingebauten kurzen Warteschleifen jeweils die beiden Farben gelb und hellbraun gegeneinander austauschen. Die im nachfolgenden dargestellte Programmschleife muß durch Drücken der STOP-Taste abgebrochen werden.

```
1000 DO
1010 SPRCOLOR 8,9
1020 FOR J=1 TO 100
1030 NEXT
1040 SPRCOLOR 9,8
1050 FOR J=1 TO 20
1060 NEXT
1070 LOOP
```

Die Verwendung von SLEEP ist nicht anzuraten, da hier mindestens eine Sekunde gewartet werden muß.

Wenn Sie nun das Programmstück starten, flattern zwar die Nachbrenner, aber es ist noch keine Bewegung im Spiel. Dies erreichen wir mittels des MOVSPR-Befehls, indem wir die Variante mit Winkel und Geschwindigkeit benutzen.

Wir bringen also Bewegung ins Spiel, indem wir nacheinander eingeben:

```
MOVSPR 1,100#20
MOVSPR 2,85#150
MOVSPR 3,90#180
MOVSPR 4,90#255
MOVXPR 5,290#60
MOVSPR 6,275#130
MOVSPR 7,270#200
MOVSPR 8,270#250
```

Damit bekommen alle Sprites eine Bewegung zugeordnet, wobei wir den Sprites #1 bis #4 eine Bewegung hauptsächlich in rechter Richtung und den Sprites #5 bis#8 eine solche in linker Richtung zugeordnet haben. Die Tiefflieger fliegen genau parallel zum unteren Bildschirmrand, was durch die Winkel 90 Grad in die eine Richtung und 270 Grad in die andere Richtung bewerkstelligt wird. Beachten Sie auch bei der Bewegungsrichtung der Sprites, daß 0 Grad nicht nach rechts geht, sondern nach oben.

Die Geschwindigkeiten wurden so gewählt, daß die größeren, im Vordergrund befindlichen Flugzeuge schneller fliegen als die hinteren, die eine längere Strecke zu bewältigen haben, während sie in unserem Blickwinkel sind.

Die Bewegung der Sprites ist Interrupt-gesteuert, so daß sie unabhängig von anderen Operationen verlaufen, wie Sie leicht feststellen können, wenn Sie sich ein Programm auslisten lassen. Allerdings arbeitet das normale Basic etwas langsamer, da es nicht mehr so viel Rechenzeit zugeteilt bekommt, die ja für die Bewegung der

Sprites verbraucht wird. Die Bewegung wird automatisch mit dem Ausschalten der Sprites angehalten, so daß dem Basic wieder die volle Rechenzeit zur Verfügung steht.

In unserem Fall ist es sinnvoll, eine kleine Programmschleife zu schreiben, in der die Laufvariable I die Werte 1 bis 8 durchläuft und innerhalb dieser Schleife die Anweisung SPRITE I,0 gegeben wird.

Es sei noch anzumerken, daß die dem Autor vorliegende Bedienungsanleitung für den Parameter n Werte zwischen 0 und 7 angegeben sind. Dies ist zu ersetzen durch die Werte 1 bis 8, da die Spritenummern im C 128-Modus eben diese Nummern besitzen (siehe auch alle anderen Sprite-Befehle). Im 64er-Modus hingegen — der jedoch keine Sprite-Befehle kennt — müssen die Nummern 0 bis 7 für die acht Sprites verwendet werden.

Wenn Sie nun zusätzlich zur Bewegung die Farbe alternieren lassen wollen, so sind die Warteschleifen in den Zeilen 1020/1030 und 1050/1070 durch die Verlangsamung des Basics unnötig geworden und können entfallen.

4/4.3.2.6

BUMP — Kollisionsüberprüfung

Etwas fehlt noch, wenn man sich die bewegenden Sprites auf dem Bildschirm ansieht. Daß die großen und kleinen Flugzeuge nicht miteinander kollidieren ist verständlich, da sie unterschiedlich weit vom Betrachter entfernt zu sein scheinen. Aber gleich große Flugzeuge müssen miteinander kollidieren, auf jeden Fall die gegnerischen (Flugzeuge der gleichen Partei könnten dicht nebeneinander fliegen).

Dazu muß man aber zunächst wissen, wann und welche Sprites miteinander kollidieren. Dieses Wissen liefert uns der BUMP-Befehl. Bevor wir jedoch zu der Besprechung des BUMP-Befehles kommen, wollen wir das bisher gezeigte in einem Programm vereinen und mit einem weiteren Detail verbessern. Hier zunächst das Listing:

```
1000 REM ----------------------------------------
1010 REM --- SPRITE-KOLLISIONEN     ---
1020 REM ---         MIT BUMP       ---
1030 REM ----------------------------------------
1040 :
1050 REM   BLOAD"TESTSPRITES"
1060 :
1070 SPRCOLOR 8,9
1080 COLOR 0,15
1090 COLOR 1,2
1100 :
1110 SPRITE 1,1,3,1,0,0,1
1120 SPRITE 2,1,3,1,1,1,1
1130 SPRITE 3,1,3,0,0,0,1
1140 GOTO 1200
1150 SPRITE 4,1,3,0,1,1,1
1160 SPRITE 5,1,12,1,0,0,1
1170 SPRITE 6,1,12,1,1,1,1
1180 SPRITE 7,1,12,0,0,0,1
1190 SPRITE 8,1,12,0,1,1,1
1200 :
1210 MOVSPR 1,0,200
1220 MOVSPR 2,30,170
1230 MOVSPR 3,100,180
1240 MOVSPR 4,80,150
1250 MOVSPR 5,300,200
1260 MOVSPR 6,340,170
1270 MOVSPR 7,240,180
1280 MOVSPR 8,280,150
1290 :
1300 MOVSPR 1,100#2
1310 MOVSPR 2,85#4
1320 MOVSPR 3,90#6
1330 MOVSPR 4,90#0
1340 MOVSPR 5,290#0
1350 MOVSPR 6,275#0
1360 MOVSPR 7,270#0
1370 MOVSPR 8,270#0
1380 :
1390 GRAPHIC 1,1
1400 :
1410 FOR I=10 TO 80 STEP 16
1420 CIRCLE 1,I,20,10,10
1430 PAINT 1,I+2,20,0
1440 NEXT
1450 :
1460 SSHAPE SS$,0,10,85,30
1470 :
1480 GSHAPE SS$,100,100,0
1490 GSHAPE SS$,150,180,0
1500 GSHAPE SS$,180,90,2
1510 GSHAPE SS$,0,150,0
1520 :
1530 BS=BUMP(1)
1540 BH=BUMP(2)
1550 :
1560 IF BS=0 AND BH=0 GOTO 1530
1570 IF BS=0 GOTO 1700
1580 :
```

Listing 4/4.3.2.6 (1)

```
1590 IF BS AND 128 THEN PRINT"SPRITE 8 - ";
1600 IF BS AND 64 THEN PRINT"SPRITE 7 - ";
1610 IF BS AND 32 THEN PRINT"SPRITE 6 - ";
1620 IF BS AND 16 THEN PRINT"SPRITE 5 - ";
1630 IF BS AND 8 THEN PRINT"SPRITE 4 - ";
1640 IF BS AND 4 THEN PRINT"SPRITE 3 - ";
1650 IF BS AND 2 THEN PRINT"SPRITE 2 - ";
1660 IF BS AND 1 THEN PRINT"SPRITE 1 - ";
1670 PRINT
1680 :
1690 IF BH=0 GOTO 1530
1700 IF BH AND 128 THEN PRINT"SPRITE 8 - HINTERGRUND"
1710 IF BH AND 64 THEN PRINT"SPRITE 7 - HINTERGRUND"
1720 IF BH AND 32 THEN PRINT"SPRITE 6 - HINTERGRUND"
1730 IF BH AND 16 THEN PRINT"SPRITE 5 - HINTERGRUND"
1740 IF BH AND 8 THEN PRINT"SPRITE 4 - HINTERGRUND"
1750 IF BH AND 4 THEN PRINT"SPRITE 3 - HINTERGRUND"
1760 IF BH AND 2 THEN PRINT"SPRITE 2 - HINTERGRUND"
1770 IF BH AND 1 THEN PRINT"SPRITE 1 - HINTERGRUND"
1780 :
1790 GOTO 1530
1800 :
1810 END
```

Listing 4/4.3.2.6 (2)

Das Einladen der Spritedaten haben wir hinter eine REM-Anweisung gesetzt, so daß sie bei Programmstart nicht jedesmal geladen wird. Dann werden die beiden Zusatzfarben für das Sprite festgelegt, sowie der Hintergrund auf hellblau geschaltet. Die Vordergrundfarbe der Grafik wird auf weiß geschaltet, die wir später noch brauchen werden,

Ab Zeile 1110 befinden sich die SPRITE-Befehle, wie wir sie weiter oben angeführt haben. Innerhalb dieser Befehle wird noch ein Sprung auf die nächste Zeile nach den Sprite-Befehlen durchgeführt, durch dessen Verschiebung innerhalb der Zeilen 1100 und 1999 Sie beliebig viele Sprites einschalten können. Dies dient dazu, den Ablauf bei Kollisionen nachher deutlicher zu machen, da sich acht bewegte Sprites zu häufig treffen, um das Geschehen am Bildschirm nachzuvollziehen.

Das Löschen der Sprites kann in einer Programmschleife erfolgen, die insgesamt auf eine der Funktionstasten gelegt werden kann. Hier ein Beispiel:

```
KEY 8,"FORI=1TO8:SPRITEI,0:NEXT"+CHR$(13)
```

Sofern Sie im Puffer für die Funktionstasten nicht genügend Platz haben, können Sie die Befehle ebenso abkürzen (FOR:F SHIFT-0 / NEXT:N SHIFT-E / SPRITE:S SHIFT-P).

Durch Drücken der Funktionstaste F8 werden dann alle Sprites gelöscht, wovon Sie sich selbst überzeugen können.

Die Positionierung der Sprites ab Zeile 1210 geschieht ebenfalls analog zu der bereits erwähnten Aufführung. Die Positionierung ist auch dann wichtig, wenn die Sprites bewegt werden sollen, da sich dadurch die Position des Bewegungsablaufes aufgrund der Anfangsposition beeinflussen läßt. Besonders deutlich wird dies an den sich waagerecht bewegenden Sprites.

Die Geschwindigkeiten wurden ab Zeile 1300 für die Sprites #4 bis #8 auf Null zurückgesetzt und für die Sprites #1 bis #4 sehr stark vermindert. Auch dies dient dazu, die folgenden Programmstücke zur Kollisionsabfrage leichter verständlich zu machen.

Neu sind die Zeilen ab 1390, wo zunächst die hochauflösende Grafik eingeschaltet und der Bildschirm gelöscht wird.

Dann wird in der FOR ...NEXT-Schleife von Zeile 1410 bis 1440 eine Folge aus lauter ausgefüllten Kreisen gezeichnet. Diese Wolke wird dann mit dem SSHAPE-Befehl in Zeile 1460 in die Variable SS$ übernommen und durch die vier GSHAPE-Befehle ab Zeile 1480 an verschiedenen Stellen am Bildschirm positioniert. Achten Sie bitte auf den Modus '2' in Zeile 1500, da sich sonst eine Ecke in einer Wolke ergeben würde.

Die Zeilen ab 1530 widmen sich dann dem BUMP-Befehl. Wie Sie im Handbuch nachlesen können, kann das Argument beim BUMP-Befehl zwei Werte annehmen, je nachdem, ob Sie eine Sprite-Sprite-Kollision oder eine Sprite-Hintergrund-Kollision abfragen wollen. Die Beschreibung in dem uns vorliegendem Handbuch ist etwas unglücklich und unvollständig, da sie die Vorgehensweise bei der Sprite-Sprite-Kollision nur ungenügend beschreibt. Deutlicher wird dies, wenn man die Verfahrensweise vom 64er-Modus kennt. Im Anhang E des Handbuches sind die Register der Sprites aufgeführt und unter den Registernummern 30 und 31 finden Sie die beiden Register für Sprite-Sprite-Kollision und Sprite-Hintergrund-Kollision. Maschinenspracheprogrammierer müssen sowieso auf diese Register zurückgreifen.

Bei einem BUMP-Befehl wird wahlweise eines der beiden Register ausgelesen. Dabei beinhaltet das Register beim Auslesen einer Sprite-Hintergrund-Kollision alle Spritenummern, die im Moment des Abfragens Berührung mit einem Zeichen des Hintergrundes haben. Da jedem Bit in diesem Register ein Sprite zugeordnet ist (Sprite #8 das am weitesten links stehende Bit (= dezimal 128), Sprite #1 das am weitesten rechts stehende Bit) ist eine Weiterbehandlung aufgrund der Abfrage im Programm ohne weiteres möglich.

Diese Behandlung finden Sie in unserem Programm ab Zeile 1700, wo wir bei einer Kollision lediglich eine Meldung an den Anwender herausgeben. Die Programmierung eines optischen und akustischen Programmteils nach einer Kollision soll Ihnen überlassen bleiben.

Das Programm ist ab Zeile 1560 so gestaltet, daß sofort zur erneuten Abfrage übergegangen wird, wenn weder eine Sprite-Sprite-Kollision noch eine Sprite-

Hintergrund-Kollision vorliegt. Liegt auch keine Sprite-Sprite-Kollision vor, so wird in Zeile 1570 gleich zur Behandlung der Sprite-Hintergrund-Kollision übergegangen. Ebenso wird nach den Anzeigen bei der Sprite-Sprite-Kollision der Programmteil ab Zeile 1700 übergangen, wenn keine Sprite-Hintergrund-Kollision vorliegt.

Sprite-Sprite-Kollisionen sind etwas schwieriger zu handhaben, da innerhalb des Registers immer mindestens zwei Bits gesetzt sind, wenn eine Kollision vorliegt, nämlich die beiden Bits der an der Kollision beteiligten Sprites. Würden hier alle acht Sprites eingeschaltet und sich mit der ursprünglich vorgegebenen Richtung und Geschwindigkeit bewegen lassen, so würde der Fall, daß alle acht Bits im Sprite-Sprite-Kollisionsregister gesetzt sind, sehr häufig auftreten. Da aber in diesem Fall nicht jedes Sprite mit jedem Sprite kollidiert ist, muß der Anwender aufgrund der Position der einzelnen Sprites selbst eine Auswahl treffen, welches Sprite mit welchem Sprite in Verbindung steht.

Aus diesem Grund haben wir in den Zeilen 1590 bis 1660 auch jeweils ein ';' ans Zeilenende gesetzt, so daß alle Kollisionen innerhalb einer Zeile dargestellt werden. Die Koordinaten eines jeden Sprites können mittels des RSPPOS-Befehls vom Rechner abgefragt werden, wie wir im letzten Unterkapitel dieses Kurses noch aufzeigen werden.

In unserem Beispielprogramm werden die Zeilen ab 1530 fortlaufend aufgerufen, so daß das Programm keine anderen Tätigkeiten mehr unternimmt, als die Kollisionsprüfung. Sollen jedoch auch noch andere Funktionen wahrgenommen werden (Textausgabe, Steuerung von Sprites mit dem Joystick), so wäre eine aufwendige Programmierung erforderlich, wenn man den COLLISION-Befehl nicht zu Hilfe nimmt.

4/4.3.2.7

COLLISION — Kollisionsprüfung mit Unterprogramm

Zur Darstellung der Wirkungsweise des COLLISION-Befehls haben wir in unserem letztgenannten Programm lediglich die Zeilen 1522 bis 1527 eingefügt und in Zeile 1790 den Sprungbefehl in ein RETURN — umgewandelt.

```
1522 COLLISION 1,1530
1524 COLLISION 2,1530
1525 :
1526 PRINT"KEINE KOLLOSION"
1527 GOTO 1526
1530 BS=BUMP(1)
1540 BH=BUMP(2)
1550 :
1560 IF BS=0 AND BH=0 GOTO 1530
1570 IF BS=0 GOTO 1700
1580 :
1590 IF BS AND 128 THEN PRINT"SPRITE 8 - ";
1600 IF BS AND 64 THEN PRINT"SPRITE 7 - ";
1610 IF BS AND 32 THEN PRINT"SPRITE 6 - ";
1620 IF BS AND 16 THEN PRINT"SPRITE 5 - ";
1630 IF BS AND 8 THEN PRINT"SPRITE 4 - ";
1640 IF BS AND 4 THEN PRINT"SPRITE 3 - ";
1650 IF BS AND 2 THEN PRINT"SPRITE 2 - ";
1660 IF BS AND 1 THEN PRINT"SPRITE 1 - ";
1670 PRINT
1680 :
1690 IF BH=0 GOTO 1530
1700 IF BH AND 128 THEN PRINT"SPRITE 8 - HINTERGRUND"
1710 IF BH AND 64 THEN PRINT"SPRITE 7 - HINTERGRUND"
1720 IF BH AND 32 THEN PRINT"SPRITE 6 - HINTERGRUND"
1730 IF BH AND 16 THEN PRINT"SPRITE 5 - HINTERGRUND"
1740 IF BH AND 8 THEN PRINT"SPRITE 4 - HINTERGRUND"
1750 IF BH AND 4 THEN PRINT"SPRITE 3 - HINTERGRUND"
1760 IF BH AND 2 THEN PRINT"SPRITE 2 - HINTERGRUND"
1770 IF BH AND 1 THEN PRINT"SPRITE 1 - HINTERGRUND"
1780 :
1790 RETURN
1800 :
1810 END
```

Listing 4/4.3.2

Die Zeilen 1526 und 1527 bilden eine prinzipielle Endlosschleife. Normalerweise würde das Programm nur die Meldung 'KEINE KOLLISION' fortlaufend ausdrucken. Da jedoch in den Zeilen 1522 und 1524 der COLLISION-Befehl verwendet wurde, wird zur Zeile 1530 verzweigt, wenn entweder eine Sprite-Sprite-Kollision oder Sprite-Hintergrund-Kollision vorliegt. In unserem speziellen Beispiel verzweigen wir in beiden Fällen zur gleichen Zeile, wo die kollidierenden Sprites mit Hilfe des BUMP-Befehls abgefragt werden.

Die Wirkungsweise des Programms veranschaulicht man sich am besten durch einen Programmstart. Sofern keine Kollision vorliegt, wird die Meldung aus Zeile 1526 ausgedruckt. Sobald jedoch entweder eine Sprite-Sprite-Kollision oder eine Sprite-Hintergrund-Kollision vorliegt, wird das Unterprogramm ab Zeile 1530 aufgerufen, wie durch die ausgegebenen Bildschirmmeldungen deutlich wird.

Beachten Sie jedoch sowohl bei der Programmierung ausschließlich mit BUMP als auch bei einem Unterprogrammsprung mittels COLLISION, daß sich während der

Programmbehandlung weitere Kollisionen ergeben können, die nicht abgeprüft werden. Dies ist besonders bei sich schnell bewegenden Sprites der Fall, und aus diesem Grunde sollte man innerhalb der Kollisionsüberprüfung und der darauffolgenden Programmbehandlung auf Einsparung von Rechenzeit bedacht sein. Hierzu sei nochmals darauf hingewiesen, daß die Bewegung der Sprites und auch Töne und Geräusche Interruptgesteuert sind und somit das Basic verlangsamen.

4/4.3.2.8

RSPCOLOR, RSPPOS und RSPRITE — Abfragebefehle für Sprites

Ebenso wie für die Parameter der hochauflösenden Grafik gibt es auch zum Thema Sprites drei Abfragebefehle, mit denen die verschiedenen Registerinhalte geprüft werden können. Da die meisten Daten jedoch für alle acht Sprites unterschiedlich sind, lohnt es sich hier, ein kleines Programm zu schreiben, mit dem alle Registerinhalte gleichzeitig auf dem Bildschirm dargestellt werden.

```
1000 REM --------------------------------
1010 REM --- ANZEIGEN VON SPRITE-    ---
1020 REM ---                DATEN    ---
1030 REM --------------------------------
1040 :
1045 PRINT"□"
1050 PRINT"BEZEICHNUNG  SPR.1   SPR.2   SPR.3   SPR.4   SPR.5   SPR.6   SPR.7
SPR.8
1060 PRINT"------------------------------------------------------------------------
------
1070 :
1080 DIM BE$(5)
1090 :
1100 DATA AKTIV,FARBE,PRIOR.,XDEHN,YDEHN,MFM
1110 :
1120 FOR I=0 TO 5
1130 READ BE$(I)
1140 NEXT
1150 :
1160 FOR I=0 TO 5
1170 PRINT BE$(I),
1175 PRINT"     ";
1180 :
1190 :FOR J=1 TO 8
1200 :PRINT RSPRITE(J,I);"      "
1210 :NEXT
1220 :
```

Listing 4/4.3.2 (1)

```
1230 PRINT
1240 NEXT
1250 :
1260 PRINT"X-POSITION    ";
1270 :
1280 FOR I=1 TO 8
1290 PRINT USING "####"; RSPPOS(I,0);
1291 PRINT"     ";
1300 NEXT
1310 :
1320 PRINT
1330 :
1340 PRINT"Y-POSITION    ";
1350 :
1355 FOR I=1 TO 8
1360 PRINT USING "####"; RSPPOS(I,1);
1365 PRINT"     ";
1370 NEXT
1375 :
1380 PRINT
1385 :
1390 PRINT"GESCHWINDIG.";
1400 :
1410 FOR I=1 TO 8
1420 PRINT USING "####"; RSPPOS(I,2);
1430 PRINT"     ";
1440 NEXT
1450
1460 PRINT
1500 PRINT"----------------------------------------------------------------------
-----
1510 :
1520 PRINT"ZUSATZFARBE 1:";RSPCOLOR (1)
1530 PRINT"ZUSATZFARBE 2:";RSPCOLOR (2)
1540 :
1550 PRINT"----------------------------------------------------------------------
-----
1560 PRINT"[illegible]"
1570 GOTO 1160
```

Listing 4/4.3.2 (2)

Besonders bei Sprite-Sprite-Kollisionen müssen die Koordinaten aller beteiligten Sprites vom Rechner abgefragt werden. Hier nützt auch kein Mitführen einer programmeigenen Variablen, da aufgrund des MOVSPR-Befehls eine programmunabhängige Bewegung von Sprites durchgeführt werden kann.

Der Aufbau des Programms wurde so gewählt, daß für jedes Sprite eine Spalte der Anzeige herangezogen wird. Parameter mit identischer Bedeutung stehen also immer innerhalb einer Zeile.

In PRINT-Befehlen in den Zeilen 1050, 1060, 1500 und 1550 muß der PRINT-Befehl durch ein '?' abgekürzt werden, um diese Zeile darstellen zu können. Der Einfachheit halber wird in Zeile 1080 ein Feld BE$ für die Bezeichnungen dimensioniert, die anschließend definiert und eingelesen werden.

In den Zeilen 1290, 1360 und 1420 wurde der PRINT-Befehl mit der Ergänzung USING benutzt, um für die unterschiedlichen Zahlenangaben jeweils eine fest definierbare Bildschirmstelle zur Verfügung zu stellen. Dieses USING kann auch in Zeile 1200 verwendet werden, falls Ihrerseits Bedarf besteht.

4/4.4

Sound beim C 128 PC

Der C 128 hat die gleichen Sound-Möglichkeiten wie der C 64. Im Gegensatz zum C 64 sind beim C 128 aber ausreichend Basic-Befehle vorhanden, die bei der Sound-Programmierung genutzt werden können. Beim 64er war man entweder auf umfangreiche POKErei angewiesen oder auf entsprechende Basic-Ergänzungen, die es ja schon nach einiger Zeit auf dem Markt gab. In Anlehnung an die wohl am häufigsten eingesetzte Basic-Erweiterung sind wohl auch die Sound-Befehle des C 128 entstanden, wenn sie auch um einiges komfortabler sind.

Im vorliegenden Kapitel wollen wir uns speziell mit den Sound-Befehlen beschäftigen, wobei wir mit dem PLAY-Befehl beginnen, der uns auch gleich das Erzeugen von Musiknoten ermöglicht. Im weiteren werden wir dann die Sound-Ausgabe verfeinern.

Da der Sound sehr mannigfach gestaltet werden kann, sind bei den hier vorgestellten Basic-Befehlen natürlich auch entsprechend viele Parameter möglich, um dem Anwender freie Hand in der Gestalt seiner Musik zu lassen. Weil die Programmierung des Sounds nicht mehr über POKE-Befehle erfolgt, und somit eine umständliche Bit-Fummelei und Registersucherei erspart bleibt, dürfte die Erzeugung von Melodien auch dem Anwender gerecht werden, dem die Sucherei nach Registern und der Bedeutung der einzelnen Bits in den Registern zu umständlich war.

4/4.4.1

PLAY — spielen Sie eine Melodie

Wenn man auf Feinheiten verzichtet, bietet der PLAY-Befehl sehr vielfältige Möglichkeiten der „Melodie-Programmierung". Der Übersichtlichkeit halber wollen wir zunächst die Angaben des Handbuches nochmals vorstellen und erläutern und uns dann einer kleinen Testmelodie zuwenden.

Grundsätzlich erfolgt die Eingabe von Noten beim PLAY-Befehl in Form von Zeichen innerhalb einer Zeichenreihe. Ähnlich dem PRINT-Befehl werden die Zeichenreihen als Konstanten **oder Variablen** (steht zwar nicht im Handbuch, geht aber) nach dem PLAY-Befehl eingegeben. Ein kleiner Unterschied besteht allerdings: Zahlvariablen und Zahlkonstanten sind nicht zugelassen.

Notenwerte („weiße Tasten")

Welche Zeichen muß man nun wie eingeben, um eine gewünschte Melodie zu erhalten? Da wären zunächst die Notenwerte zu nennen. Sie werden eingegeben, wie sie normalerweise auch als Buchstaben angesprochen werden, also als: C D E F G A B (nicht H).

Die Notenschrift ist also in Buchstaben umzusetzen, wobei hier ausdrücklich darauf hingewiesen wird, daß wir zwar im Deutschen den Ton unterhalb C als H bezeichnen, dieser im englischen Sprachgebrauch aber mit B angesprochen wird. Obige Aufzählung enthält also — auf einer Klaviertastatur gesehen — alle ganzen Noten einer Oktave von C bis B (= H).

Notenwerte („schwarze Tasten")

Auch die „schwarzen Tasten" sind auf diese Weise anzusprechen. In der Musik werden sie durch ein „#" oder „b" vor der jeweiligen Note dargestellt. Das „#" bedeutet dabei eine Erhöhung um einen halben Ton und das „b" eine Verminderung um einen halben Ton. Bei Ihrem C 128 ist dem jeweiligen Buchstaben für die „weiße Taste" entsprechend ein „#" für die Erhöhung um einen halben Ton voranzustellen.

Da wir den Buchstaben B/b bereits für die — im europäischen Sprachgebrauch — Note H verwendet haben, muß für die Verminderung um einen Halbton ein anderes Zeichen herangezogen werden. Dies ist das „$".

Eine komplette Oktave mit Halbtönen als Eingabe in Ihrem Rechner würde dabei also wie folgt aussehen (je nach Vorzeichen):

C		
#C	=	$D
D		
#D	=	$E
E		
F		
#F	=	$E
G		
#G	=	$A
A		
#A	=	$B
B		

Oktaven

Damit können wir nun eine Oktave spielen, aber wie sieht es mit den Oktavunterschieden aus. Dies ist ganz einfach, hinter dem Buchstaben „O“ kann man die Nummer einer Oktave angeben. Diese Nummer kann zwischen 0 und 6 liegen, es stehen also sieben Oktaven zur Verfügung. Da Sie — wie wir später sehen werden — die Zeichen für die Noten einfach hintereinander schreiben, ist es nicht schwer, zwischendurch auch noch mit z.B. „03“ anzugeben, daß die folgenden Noten mit Tonhöhe der dritten Oktave gespielt werden.

Sie sollten sich beim Eingeben der Daten auf jeden Fall sehr konzentrieren, da ein Nachvollziehen der Noten am Bildschirm sehr mühsam ist.

Notendauer

Nun haben wir also eine elektronische Orgel, die uns die Noten von sieben Oktaven spielen kann. Für ein korrektes Notenspiel muß es allerdings auch noch die Möglichkeit geben, die Dauer jeder Note festzulegen.

Auch die Notendauern werden durch Buchstaben gekennzeichnet, wobei jeder Notendauer von Werten einer ganzen Note bis hin zu einer 1/16 Note ein Buchstabe zugeordnet ist. Nicht jeder Note im PLAY-Befehl muß eine Notendauer zugeordnet werden, vielmehr gilt die eingegebene Notendauer, bis die Notendauer durch Eingabe eines entsprechenden anderen Buchstabens verändert wird. Den Notendauern sind folgende Buchstaben zugeordnet:

W	:	ganze Note	(wahrscheinlich vom engl. whole)
H	:	halbe Note	(engl.: half)
Q	:	viertel Note	(engl.: quarter)
I	:	achtel Note	
S	:	sechzehntel Note	

Auch das Punktieren einer Note (die Notendauer wird um die Hälfte verlängert) ist möglich. Hierfür ist bei der Eingabe ebenfalls ein Punkt vorgesehen.

Pause

Auch Pausen sind bei der Eingabe möglich. Eine Pause wird durch ein „R“ angegeben. Einer Pause wird die gleiche Dauer wie einer Note (siehe oben) zugewiesen.

Klangfarben

Wie auch bei einer richtigen Orgel kann Ihr C 128 mit verschiedenen Klangfarben aufwarten. Zehn solcher Klangfarben sind bereits im Rechner vorhanden. Diese können aber auch von Ihnen geändert werden, wie wir beim ENVELOPE-Befehl später sehen werden.

Klangfarben werden ähnlich den Oktaven aufgerufen, indem nach einem „T“ (engl.: tone) die Nummer der Klangfarbe eingegeben wird. Möglich sind Eingaben von „0“ bis „9“, wobei folgende Tabelle zugrunde liegt:

0	—	Klavier
1	—	Akkordeon
2	—	Zirkusorgel
3	—	Trommel
4	—	Flöte
5	—	Gitarre
6	—	Cembalo
7	—	Orgel
8	—	Trompete
9	—	Xylophon

Einige Klangeffekte sind gut getroffen, andere sind weniger gut zu erkennen, so daß bei diesen — aber auch für eigene Anwendungen — die Verwendung des ENVELOPE-Befehls anzuraten ist.

Lautstärke

Neben den Notenwerten, den Notendauern und der Klangfarbe können Sie auch noch weitere Parameter bestimmen. Zunächst ist hier die Lautstärke zu erwähnen. Diese kann zwar generell mit dem VOL-Befehl gesetzt werden (siehe unten), aber auch die Eingabe in der Zeichenreihe bei PLAY ist möglich. Hier kann eine Lautstärke zwischen 0 und 9 ausgewählt werden, der ein „U" voranzustellen ist.

Stimmenauswahl

Wie der C 64 auch, hat der C 128 drei Stimmen, die im PLAY-Befehl mit einem „V" und der darauffolgenden Nummer zwischen 1 und 3 angewählt werden kann.

Filter

Als letztes ist noch der Filter zu erwähnen, der allerdings nur ein- bzw. ausgeschaltet werden kann. Nach einem „X" ist eine „1" (für ein) oder eine „0" (für aus) anzugeben. Die Parameter des Filters werden mit dem FILTER-Befehl bestimmt.

Noch ein kleiner Hinweis: Sofern Sie ein nicht vorgesehenes Zeichen in Ihrer Zeichenreihe eingegeben haben, wird entweder ein ILLEGAL QUANTITY ERROR ausgegeben oder das Spiel der Musik abgebrochen.

Nachdem wir nun alle Möglichkeiten beim PLAY-Befehl durchgesprochen haben, wollen wir ein kleines Beispiel starten. Als Testmelodie soll uns das sicherlich bekannte Lied Sun of Jamaika dienen.

```
100 REM --- SUN OF JAMAIKA EINSTIMMIG ---
110 :
200 PLAY"U6T9O3HFQFEHFQFGHAQAGHAQAO4CHDQDCO3HAQGFWGG"
210 PLAY"U9T1HEQEFHGQFEHFQFDWCHEQEFHGQA$BWAHAQA"
220 PLAY"O4CHDQDCO3HAQAO4CHDQDCO3HAQAO4CHDQDC"
230 PLAY"O3HAQGFWGHGQ.$BIAWGG"
```

Listing 4/4.4-1

In Zeile 200 schalten wir zunächst die Lautstärke auf den Wert „6" und anschließend suchen wir uns als Klangfarbe das Xylophon aus. Die ersten Noten sollen mit der Tonhöhe der Oktave 3 gespielt werden, und die erste Note bekommt die Länge einer halben Note zugewiesen. Dann wird ein F gespielt und anschließend die Notendauer auf eine 1/4 Note heruntergesetzt, wonach die Töne F und E gespielt werden.

Nach dem Umschalten auf die Dauer einer halben Note wird erneut ein F gespielt, wonach die Noten F und G als Viertelnoten folgen. Soweit zur Einführung, im folgenden wollen wir nun noch auf die wichtigsten Änderungen eingehen.

Zu Beginn des letzten Drittels der Zeile 200 wird die Notenhöhe auf Oktave 4 umgeschaltet und kurze Zeit später wieder auf die dritte Oktave zurückgeschaltet. Die letzten beiden Noten aus Zeile 200 (beide G) werden als ganze Note gespielt, was durch das „W" angezeigt ist.

In Zeile 210 erhöhen wir die Lautstärke auf den maximalen Wert und suchen uns als Klangfarbe das Akkordeon aus. Gegen Ende der Zeile finden Sie auch eine „schwarze Taste", nämlich das B (im europäischen Sprachgebrauch), das durch ein „$B" dargestellt wird. In Zeile 230 im letzten Drittel wird auch eine punktierte Viertelnote mit ,,Q.'' gespielt.

Sie sehen, die Eingabe von Melodien ist recht einfach, wenn auch das „Notenlesen" etwas außergewöhnlich gegenüber normalen Notenblättern ist.

Mehrstimmiges Spiel

Da wir auch zwischen drei verschiedenen Stimmen wählen können, liegt es nahe, alle drei Stimmen gleichzeitig spielen zu lassen. So kann z.B. die erste Stimme die Hauptmelodie spielen (cantus firmus) und die beiden anderen Stimmen übernehmen die Begleitung.

Einfach wäre es, wenn man jede der drei Stimmen getrennt eingeben könnte, und diese vom Rechner verbunden würden. Dazu müßte man sich allerdings ein kleines Programm schreiben.

Wenn Sie drei Stimmen gleichzeitig spielen lassen wollen, kommen Sie nicht umhin, innerhalb der Eingabe bei PLAY jeweils die Note einzeln für jede Stimme hinter-

einander zu schreiben. Im folgenden Listing haben wir dies an den ersten Takten des bekannten Liedes vorgeführt:

```
100 REM --- SUN OF JAMAIKA MEHRSTIMMIG ---
110 :
120 PLAY"V1T1U9O3HFV2U3O2QAV3U4FV2AV3FV1U9O3FV2U3O2AV3FV1U9O3EV2U3O2GV3C"
130 PLAY"V1U9O3HFV2U3QO2AV3FV2AV3FV1U9O3FV2U3O2AV3FV1U9O3GV2U3O2GV3C"
140 PLAY"V1O3U9HAV2O2U3QAV3FV2AV3FV1U9O3AV2O2U3AV3FV1U9O3GV2O2U3GV3C"
150 PLAY"V1O3U9HAV2O2U3QAV3FV2AV3FV1U9O3AV2O2U3AV3FV1U9O4CV2O2U3AV3F"
160 PLAY"V1O4U9HDV2O2U3QFV3O1BV2O2FV3O1BV1U9O4DV2O2U3FV3O1BV1U9O4CV2O2U3EV3C"
170 PLAY"V1O3U9AV2O2U3QAV3FV2AV3FV1U9O3GV2O2U3GV3CV1U9O3FV2O2U3AV3F"
```

Listing 4/4.4-2

Wie Sie an diesem kleinen Beispiel sehen, verdreifacht sich nicht der Aufwand für drei statt einer Stimme, die spielen sollen, sondern er verachtfacht sich. Dies liegt unter anderem daran, daß in der Regel die Begleitstimmen eine Oktave tiefer liegen, und somit nach jedem Umschalten zwischen den Stimmen auch die Oktave umgeschaltet werden muß. In unserem Beispiel waren es die Oktaven „O3" und „O2", wobei die dritte Stimme zeitweise auch noch in die Oktave „O1" hinabgesunken ist.

Außerdem sollen ja auch nicht die Begleitstimmen genauso laut wie die Melodiestimme erscheinen, so daß auch jeweils noch die Lautstärke (in unserem Fall (U9/U3) geändert werden muß. Vielfach sind auch die Notendauern der einzelnen Stimmen unterschiedlich, und würden Sie auch noch jeder Stimme eine eigene Klangfarbe zuordnen, so müßte auch noch die Klangfarbe umgeschaltet werden, was die Ausgabe noch weiter in die Länge zieht.

Die Tastatur als Orgel

Die Eingabe von Melodien mittels des PLAY-Befehls hat natürlich den Vorteil, daß man diese speichern und zu jedem beliebigen Zeitpunkt neu abspielen kann. Wenn man jedoch ein Stück direkt spielen will, kann man die Tastatur als Orgelklaviatur benutzen. Dies ist natürlich nur mit einem Programm möglich, das wir uns im folgenden erarbeiten wollen.

Bevor wir uns jedoch überlegen, wie das Programm aufgebaut sein soll, muß eine Entscheidung fallen, wo die Töne auf der Tastatur hervorgerufen und die „Register" der Orgel (Klangfarbe etc.) eingestellt werden sollen. Um die nötigen Tastendrucke recht einfach zu halten und weitgehend von der normalen Arbeit mit der Tastatur abzuheben, haben wir uns für das in Bild 4/4.4-1 dargestellte Schema entschieden.

Die „weißen" Tasten einer Oktave wollen wir den Tasten mit den Buchstaben Q, W, E, R, T, Y, U zuordnen und die Halbtöne dazwischen entsprechend den Tasten 2 und 3 sowie 5, 6 und 7.

Den Klang unserer Orgel stellen wir dann hauptsächlich über mit Pfeilen bezeichnete Tasten ein. So z.B. die Notendauer mit dem Pfeil nach links, wobei bei jedem Tastendruck die Notendauer verdoppelt wird. Ist der Wert einer ganzen Note erreicht, so gehen wir wieder zur Sechzehntelnote über. Die Oktavwerte sollen mit den Tasten für „Cursor nach oben" und „Cursor nach unten" gesteuert werden, wobei ein Druck auf die Taste „Cursor nach unten" den aktuellen Oktavwert um eine Einheit vermindert und entsprechend die Taste „Cursor nach oben" den Oktavwert erhöht.

Analog sollen die Tasten „Cursor nach links" und „Cursor nach rechts" für die Klangnummern herangezogen werden, wobei — leicht einprägsam — eine höhere Klangnummer durch „Cursor nach rechts" erzeugt wird. Die Taste „Cursor nach links" hat dann den gegenteiligen Effekt.

Die Lautstärke soll mit den Tasten „+" und „-" gesteuert werden, wobei „+" sinngemäß einer höheren Lautstärke entspricht und „-" einer geringeren Lautstärke. Auch bei den Oktavwerten, den Klangfarben und der Lautstärke wird der Wert jeweils um eine Einheit höher oder niedriger als der aktuelle Wert gesetzt.

Mit unserem kleinen Programm wollen wir Ihnen natürlich nur wieder eine Anregung geben. Verbesserungsmöglichkeiten gibt es bei Software naturgemäß immer, so daß Sie z.B. die Tasten rechts von U als weitere Oktave benutzen können. Achten Sie im späteren Programm jedoch darauf, daß bei der Ausgabe der Noten dann eine höhere Oktave als die angewählte, anzusprechen ist. Außerdem können Sie die Notendauer von der Zeit der gedrückten Taste abhängig machen. Wer die **O**ktav- und **K**langwerte sowie die **L**autstärke trotzdem über eine Ziffer eingeben will, kann entsprechend die Tasten K, O und L — gefolgt von einer gültigen Ziffer — zulassen, da die genannten Tasten nicht für Noten herangezogen werden.

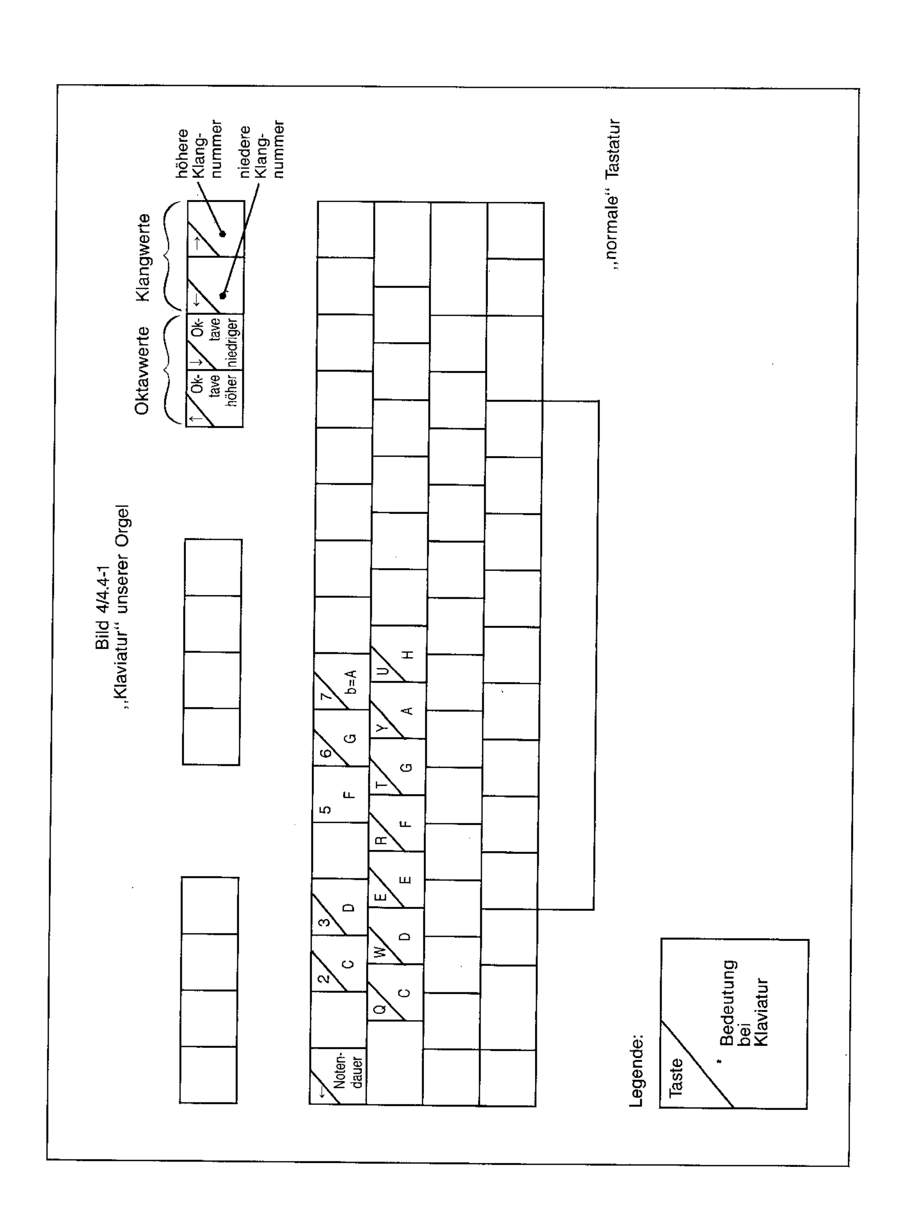


Bild 4/4.4-1
„Klaviatur" unserer Orgel

Beginnen wir also mit unserem Programm:

```
1000 REM ------------------------------------------
1010 REM ---- DIE KLAVIATUR ALS ORGEL ---
1020 REM ------------------------------------------
1030 :
1040 REM ---- FELDER FUER NOTENPARAMETER ---
1050 :
1060 DIM OK$(6)                          REM OKTAVE
1070 DATA O0,O1,O2,O3,O4,O5,O6
1080 FOR I=0 TO 6
1090 READ OK$(I)
1100 NEXT
1110 O=3
1120 O$=OK$(3)
1130 PRINT"[illegible]OKTAVE:      ";O
1140 :
1150 DIM KL$(9)                        : REM KLANG
1160 DATA T0,T1,T2,T3,T4,T5,T6,T7,T8,T9
1170 FOR I=0 TO 9
1180 READ KL$(I)
1190 NEXT
1200 DIMK$(9)
1210 DATA KLAVIER,AKKORDEON,ZIRKUSORGEL,TROMMEL,FLOETE,GITARRE,CEMBALO,ORGEL
1220 DATA TROMPETE,XYLOPHON
1230 FOR I=0 TO 9
1240 READ K$(I)
1250 NEXT
1260 K=7
1270 O$=O$+KL$(K)
1280 PRINT"KLANG:       ";K;"    ";K$(K)
1290 :
1300 DIM ND$(5),DA$(5)                 : REM DAUER
1310 DATA GANZE,W,HALBE,H,VIERTEL,Q,ACHTEL,I,SECHZEHNTEL,S
1320 FOR I=1 TO 5
1330 READ ND$(I)
1340 READ DA$(I)
1350 NEXT
1360 D=2
1370 D$=ND$(D)
1380 PRINT"DAUER:        ";DA$(D);"     ";ND$(D);" NOTE"
1390 :
1400 DIM LT$(9)                   : REM LAUTSTAERKE
1410 DATA U0,U1,U2,U3,U4,U5,U6,U7,U8,U9
1420 FOR I=0 TO 9
1430 READ LT$(I)
1440 NEXT
1450 L=6
1460 L$=LT$(L)
1470 PRINT"LAUTST.:     ";L
1480 :
2000 REM ---- ABFRAGESCHLEIFE           ---
2010 :
2020 REM ---- NOTENWERTE                ---
2030 :
2040 GETKEY A$
2050 IF A$="Q" THEN PLAY O$+"C"   : GOTO 2040
2060 IF A$="2" THEN PLAY O$+"#C"  : GOTO 2040
2070 IF A$="W" THEN PLAY O$+"D"   : GOTO 2040
```

Listing 4/4.4-3 (1)

```
2080 IF A$="3" THEN PLAY O$+"#D"  : GOTO 2040
2090 IF A$="E" THEN PLAY O$+"E"   : GOTO 2040
2100 IF A$="R" THEN PLAY O$+"F"   : GOTO 2040
2110 IF A$="5" THEN PLAY O$+"#F"  : GOTO 2040
2120 IF A$="T" THEN PLAY O$+"G"   : GOTO 2040
2130 IF A$="6" THEN PLAY O$+"#G"  : GOTO 2040
2140 IF A$="Y" THEN PLAY O$+"A"   : GOTO 2040
2150 IF A$="7" THEN PLAY O$+"#A"  : GOTO 2040
2160 IF A$="U" THEN PLAY O$+"B"   : GOTO 2040
2170 :
2180 REM --- OKTAVWERTE AENDERN        ---
2190 :
2200 IF ASC(A$)=145 THEN IF O<6 THEN GOSUB 3000 : GOTO 2040
2210 IF ASC(A$)=17  THEN IF O>0 THEN GOSUB 3500 : GOTO 2040
2220 :
2230 REM --- KLANGWERTE AENDERN         ---
2240 :
2250 IF ASC(A$)=29  THEN IF K<9 THEN GOSUB 4000 : GOTO 2040
2260 IF ASC(A$)=157 THEN IF K>0 THEN GOSUB 4500 : GOTO 2040
2270 :
2280 REM --- NOTENDAUERN AENDERN        ---
2290 :
2300 IF A$="←"  THEN GOSUB 5000 : GOTO 2040
2310 :
2320 REM --- LAUTSTAERKE AENDERN        ---
2330 :
2340 IF A$="+" THEN IF L<9 THEN GOSUB 6000 : GOTO 2040
2350 IF A$="-" THEN IF L>0 THEN GOSUB 6070 : GOTO 2040
2360 :
2370 GOTO 2040
2380 :
3000 REM --- OKTAVE ERHOEHEN        ---
3010 :
3020 O=O+1
3030 O$=OK$(O)+MID$(O$,3)
3040 PRINT"[illegible]OKTAVE:     ";O
3050 RETURN
3060 :
3500 REM --- OKTAVE VERMINDERN        ---
3510 :
3520 O=O-1
3530 O$=OK$(O)+MID$(O$,3)
3540 PRINT"[illegible]OKTAVE:     ";O
3550 RETURN
3560 :
4000 REM --- KLANG ERHOEHEN        ---
4010 :
4020 K=K+1
4030 O$=LEFT$(O$,3)+KL$(K)+MID$(O$,5)
4040 PRINT"[illegible]KLANG:       ";K;"     ";K$(K);"                 "
4050 RETURN
4060 :
4500 REM --- KLANG ERHOEHEN        ---
4510 :
4520 K=K-1
4530 O$=LEFT$(O$,2)+KL$(K)+MID$(O$,5)
4540 PRINT"[illegible]KLANG:       ";K;"     ";K$(K);"                 "
4550 RETURN
4560 :
```

Listing 4/4.4-3 (2)

```
5000 REM --- DAUER AENDERN         ---
5010 :
5020 D=D-1
5030 IF D=0 THEN D=5
5040 O$=LEFT$(O$,4)+DA$(D)+MID$(O$,6)
5050 PRINT"SQQQQQQQQDAUER:        ";DA$(D);"      ";ND$(D);" NOTE         "
5060 RETURN
5070 :
6000 REM --- LAUTSTAERKE ERHOEHEN    ---
6010 :
6020 L=L+1
6030 O$=LEFT$(O$,5)+LT$(L)+MID$(O$,8)
6040 PRINT"SQQQQQQQQQLAUTST:     ";L
6050 RETURN
6060 :
6070 REM --- LAUTSTAERKE VERMINDERN  ---
6080 :
6090 L=L-1
6100 O$=LEFT$(O$,5)+LT$(L)+MID$(O$,8)
6110 PRINT"SQQQQQQQQQLAUTST:     ";L
6120 RETURN
```

Listing 4/4.4-3 (3)

Da wir nach jedem Tastendruck dem Computer die komplette Information über die zu spielende Note übermitteln wollen (Lautstärke, Oktave, Klangfarbe etc.), ist es sinnvoll, diese zuvor als Zeichenreihen zu definieren. Durch diese Vorgehensweise ist Ihnen die Möglichkeit gegeben, das Programm später nach Belieben zu ändern, speziell zu erweitern.

Felder für Noten-Parameter

Zu Beginn des Programms werden also einige Felder definiert, die nur den „Text" für den PLAY-Befehl an den Computer übernehmen. Über die Indices der Felder können dann die einzelnen Parameter zur Note gesteuert werden. Analog zu den Indexnummern wird also die jeweilige Oktavhöhe, der Klangwert oder die Lautstärke angegeben. Außerdem sollen die aktuell eingestellten Werte am Bildschirm erkennbar sein.

Wir beginnen also in Zeile 1060 mit der Definition des Feldes für die Oktavwerte, die anschließend in Zeile 1070 in einer DATA-Zeile dargestellt sind und ab Zeile 1080 in das Feld OK$() eingelesen werden. Die Voreinstellung für die Oktave (Zeile 1110) beträgt 3, und unseren Variablen für die aktuelle Oktavhöhe (0) wird dieser Wert in Zeile 1120 zugewiesen, wonach noch in Zeile 1130 der aktuelle Wert am Bildschirm ausgegeben wird.

Genau wie bei den Oktavwerten werden die Klangwerte besetzt, wobei jedoch einige Änderungen zu berücksichtigen sind. Nach der Definition des Feldes, den Daten und dem Einlesen der Daten in das Feld KL$() wird noch ein zusätzliches Feld K$() mit der gleichen Länge wie KL$() definiert, in dem die Namen der einzelnen Klangnummern festgehalten sind. Auch diese werden ab Zeile 1230 eingelesen. Die Vorbesetzung der Klangfarbe: Die Orgel mit Klangnummer 7.

Bevor die Klangnummer und der zugehörige Text in Zeile 1280 ausgegeben werden, wird der Variablen, die bisher den Oktavwert enthält,O$ noch die Klangfarbe hinzugefügt.

Ähnlich wie bei der Oktavhöhe und der Klangfarbe wird bei der Notendauer vorgegangen, wobei hier die beiden Felder ND$() für die Bezeichnung der Notendauer und DA$() für die Computereingabe der Notendauer innerhalb einer Zeile definiert werden (Definitionen mehrerer Felder innerhalb einer DIM-Anweisung ist möglich, wenn die einzelnen Felder durch Kommata getrennt sind). Die Daten für die Felder ND$() und DA$() wurden abwechselnd in der DATA-Zeile 1310 eingegeben, so daß sie auch innerhalb einer einzigen Schleife ab Zeile 1320 eingelesen werden können. Die Voreinstellung entspricht dem Wert für eine halbe Note. Als Abschluß wird wieder die entsprechende Ausgabe auf dem Bildschirm getätigt.

Als letzte Vorgabe wird analog zu obiger Vorgehensweise die Lautstärkeninformation für den Computer aufbereitet, worauf wir jedoch nicht mehr näher eingehen wollen.

Wichtig für den Ablauf sind besonders folgende Variablen:

O :	Aktuelle Oktavhöhe
K :	Aktuelle Klangnummer
D :	Aktuelle Notendauer
L :	Aktuelle Lautstärke
OK$():	Text für PLAY-Befehl — Oktavhöhen
KL$() :	Text für PLAY-Befehl — Klangnummer
K$() :	Bezeichnung der Klangfarben
DA$()	Text für PLAY-Befehl — Notendauer
NS$() :	Bezeichnung der Notendauern
LT$() :	Text für PLAY-Befehl — Lautstärke

Abfrageschleife

In der nun folgenden Zeilengruppe ab Zeile 2040 werden fortlaufend die Eingaben von der Tastatur abgefragt. Liegt kein gültiges Zeichen vor, so wird in Zeile 2370 auf den Abfragebefehl in Zeile 2040 zurückgesprungen.

Notenwerte

Ab Zeile 2050 werden die Tasten für die einzelnen Noten (siehe Bild 4/4.4-1) in entsprechende Anweisungen für den PLAY-Befehl umgewandelt. Dabei stellt O$ die Zeichenreihe für den PLAY-Befehl dar, und die Note wird an die anderen Eingaben angehängt. Da jeweils nur eine Taste gedrückt werden kann, kann sofort wieder zur Abfrage in Zeile 2040 übergegangen werden, was erhebliche Rechenzeitvorteile bringt.

Oktavwerte ändern

Liegt kein Tastendruck für eine gültige Note vor, so wird in den Zeilen 2200 und 2210 abgefragt, ob die Oktavhöhe verändert werden soll. Die Abfrage erfolgt über die ASC-Funktion, die mit dem Code für die Tasten „Cursor nach oben" und „Cursor nach unten" verglichen wird. Die entsprechenden Unterprogramme ab Zeile 3000 bzw. 3500 werden nur angesprungen, wenn durch den Tastendruck der gültige Bereich für die Oktaven (0-6) nicht über- bzw. unterschritten wird.

Klangwerte ändern

Die beiden Zeilen 2250 und 2260 arbeiten analog zu den Zeilen 2200 und 2210, jedoch werden hier die Tastencodes 29 und 157 abgefragt, die den Tasten „Cursor nach links" und „Cursor nach rechts" zugeordnet sind. Auch hier wird geprüft, ob der gültige Bereich verlassen würde, wenn die Unterprogramme ab den Zeilen 4000 bzw. 4500 aufgerufen würden.

An dieser Stelle noch etwas zu der Vorgehensweise bei den IF-Abfragen: Natürlich ist es auch möglich, z.B. Zeile 2250 wie folgt zu formulieren: „IF ASC(A$)=29 AND K<9 THEN . . . " Die Abfrage über die Gültigkeit des Bereiches wird in unserem Fall aber nur erreicht, wenn überhaupt der richtige Tastendruck vorlag, was erhebliche Rechenzeit spart, da die weiteren Zeilen früher erreicht werden, weil die UND-Verknüpfung nicht durchgeführt werden muß.

Notendauer ändern

Die Notendauer wird gemäß unserer Konvention mit der Taste „Pfeil nach links" geändert. Da die Gültigkeit des Bereiches im Unterprogramm überprüft wird (Übergang von ganzer Note auf Sechzehntelnote), braucht an dieser Stelle keine Plausibilitätsprüfung durchgeführt zu werden.

Lautstärke ändern

Als letztes wird geprüft, ob die Tasten zur Veränderung der Lautstärke gedrückt wurden. Die Vorgehensweise ist analog den Zeilen 2200/2210 bzw. 2250/2260.

Unterprogramm: Oktave erhöhen

Die Unterprogramme ab Zeile 3000 werden nur erreicht, wenn die entsprechende Taste gedrückt wurde und der Bereich bei einem Aufruf nicht über- bzw. unterschritten wird. In den Unterprogrammen brauchen wir uns also nur noch auf die Durchführung der veranlaßten Tätigkeit zu konzentrieren.

Zunächst wird in dem Unterprogramm unser Wert für die aktuelle Oktave (O) erhöht. Da in O$ der gesamte Text für den PLAY-Befehl vorliegt, müssen wir selektiv den Teil ändern, der die Oktavhöhe betrifft. Bild 4/4.4-2 zeigt das Schema, das wir für die Variable O$ zugrunde gelegt haben. Dabei sind die ersten beiden Zeichen innerhalb der Zeichenreihenvariable den Oktavwerten vorbehalten.

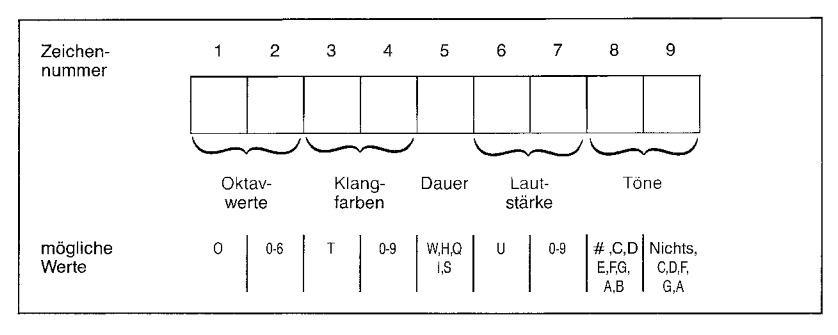


Bild 4/4.4-2 Schema für Aufbau der Variablen O$

Sofern Sie das Programm nicht verändern, beinhaltet das erste Zeichen von O$ immer ein „O“, und das zweite Zeichen die jeweilige Höhe mit den Werten „0“ bis „6“. Im Feld OK$() ist dem Index entsprechend eine Buchstabengruppe aus zwei Buchstaben bestehend vorbereitet (siehe Definition der Felder). Über unseren aktuellen Zeiger O greifen wir also das für uns richtige Element aus dem Feld heraus und positionieren es zu Beginn des Strings O$. Der Rest von O$ wird mit dem MID$()-Befehl unverändert übernommen.

An dieser Stelle sei vermerkt, daß beim NID$()-Befehl auch der letzte Parameter weggelassen werden kann, der die Länge der zu „mittelnden“ Zeichenreihe angibt. In diesem Fall wird der Rest der Zeichenreihe herangezogen. Die Verwendung des MID$()-Befehls enthebt uns einiger Rechenarbeit, die beim RIGHT$()-Befehl gegeben wären.

Bevor wir das Unterprogramm verlassen, wollen wir noch den aktuellen Oktavwert am Bildschirm anzeigen. Dafür wird zunächst der Cursor in die linke obere Ecke des Bildschirms gebracht (inverses S für HOME) und anschließend um fünf Zeilen nach unten bewegt.

Unterprogramm: Oktave vermindern

Außer Zeile 3520, in der eine Einheit von der Variablen O abgezogen wird, ergibt sich keine Änderung zu vorgenanntem Unterprogramm.

Unterprogramm: Klang erhöhen

Auch dieses Programmstück ist analog dem Unterprogramm ab Zeile 3000 aufgebaut. Statt der Variablen O wird jetzt jedoch der aktuelle Wert für die Klangnummer (K) herangezogen. Außerdem ist der Wert für den PLAY-Befehl in der Variablen O$ im dritten und vierten Zeichen untergebracht, so daß man zunächst die beiden linken Zeichen und alle Zeichen ab dem fünften nach rechts übernehmen muß.

In Zeile 4030 wird der entsprechende Wert mitten in den Wert von O$ eingebaut. Auch hier folgt wieder die Ausgabe auf dem Bildschirm, zunächst die neue Nummer, anschließend die zugehörige Bezeichnung der Klangfarbe.

Unterprogramm: Klang erhöhen

Analog letztem Unterprogramm, lediglich in Zeile 4520, ändert sich die Rechenart („-“ statt „+“).

Unterprogramm: Dauer ändern

Auch das Unterprogramm ab Zeile 5000 ist in der Struktur aufgebaut wie die bisher beschriebenen. Beachten Sie jedoch, daß es zum Ändern der Notendauer nur ein einziges Unterprogramm gibt, und die Notendauer immer verringert wird. In Zeile 5030 wird der Sprung von ganzer Note auf sechzehntel Note durchgeführt, da dem Wert 0 in D keine Daten in den Feldern DA$() und ND$() zugewiesen sind (die Programmschleife in Zeile 1320 beginnt mit dem Wert „1“). Dem Buchstaben für die Notendauer ist in unserer Zeichenreihenvariablen O$ die fünfte Stelle vorgesehen, so daß die ersten vier Zeichen und alle Zeichen ab dem sechsten unverändert übernommen werden.

Unterprogramm: Lautstärke erhöhen

Auch hier treffen wir wieder die analoge Vorgehensweise, wobei die Variable L die aktuelle Lautstärke angibt und natürlich in O$ das sechste und siebte Zeichen zur Übergabe an den Rechner beim PLAY-Befehl herangezogen werden.

Unterprogramm: Lautstärke vermindern

Siehe Lautstärke erhöhen; Subtraktion in Zeile 6090.

Damit sind die grundlegenden Möglichkeiten eines Orgelprogrammes gegeben. Wie bereits erwähnt, steht nichts im Wege, daß Sie das Programm Ihren Wünschen anpassen und verbessern. Bisher haben wir von den Sound-Befehlen des C 128 nur den PLAY-Befehl kennengelernt. Dies wollen wir nun ändern und zum nächsten wichtigen Befehl übergehen.

4/4.4.2

ENVELOPE — Der Klang macht die Musik

Mit dem PLAY-Befehl haben wir schon den wirkungsvollsten und mächtigsten Befehl kennengelernt. Durch seine hohe Zahl von Parametern ist es möglich, fast jeden beliebigen Klang zu erzeugen. Dabei ist man jedoch nicht auf die schon verwendeten vorgegebenen Klangfarben festgelegt. Dem einen oder anderen wird diese oder jene Klangfarbe nicht gefallen oder er wünscht sich etwas ausgefalleneres. Der ENVELOPE-Befehl macht dies möglich.

Auch der ENVELOPE-Befehl hat eine Reihe von Parametern und weist damit eine ähnliche Mächtigkeit auf wie der PLAY-Befehl. Umsteigern von C 64 wird auffallen, daß hier die tonbeeinflussenden Parameter des C 64 (Attack, Decay, Sustain und Release) sowie die Wellenform und natürlich die Impulsbreite (bei Rechteck-Schwingung) mit einem einzigen Befehl manipuliert werden können, wobei der Befehl auch noch komfortabler ist als derjenige aus Simon's Basic.

Das Format innerhalb der Bedienungsanleitung sieht vielleicht etwas kompliziert aus, ist es aber nicht. Die vielen Klammern sagen lediglich aus, daß die Parameter in beliebiger Weise weggelassen werden können, jedoch ist für einen weggelassenen Parameter jeweils das Komma zu setzen. Wollen Sie also die Implusbreite auf 2000 setzen, so können Sie eingeben

```
ENVELOPE 1,,,,,,2000
```

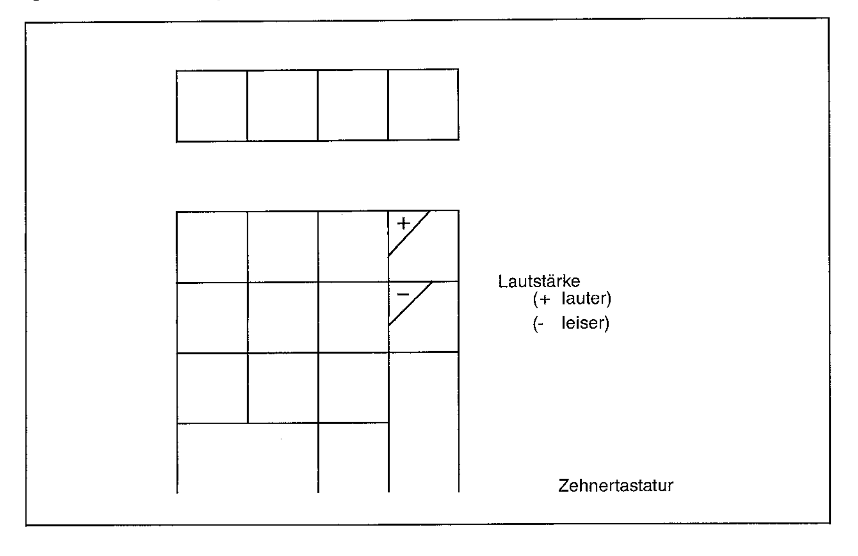
dadurch behalten die anderen Parameter ihren Wert bei. Die Eingabe von

ENVELOPE 1,2000

würde zu einer unerlaubten Eingabe führen, da der Computer schließlich nicht wissen kann, welche der Parameter Sie meinen, wenn die Kommata nicht angeführt sind. Also: Anhand der Zahl der Kommata weiß der Rechner, welchen Wert Sie ändern wollen.

Noch einige Hinweise zum Befehl: Umsteigern vom C 64 wird die Bedeutung der einzelnen Parameter schon bekannt sein. Trotzdem wollen wir im folgenden noch einmal kurz auf ihre Bedeutung eingehen. Bild 4/4.4-3 veranschaulicht die Zusammenhänge der vier Parameter Anschlagzeit, Abschwellzeit, Haltezeit und Ausklingzeit. Wie beim 64er auch, werden diese jeweils in einem Halbbyte gespeichert, jedoch braucht sich beim C 128 kein Anwender mehr darum zu kümmern, es sei

denn, er programmiert Maschinensprache. Der ENVELOPE-Befehl erspart ganz speziell eine umfangreiche Bit-Fummelei.



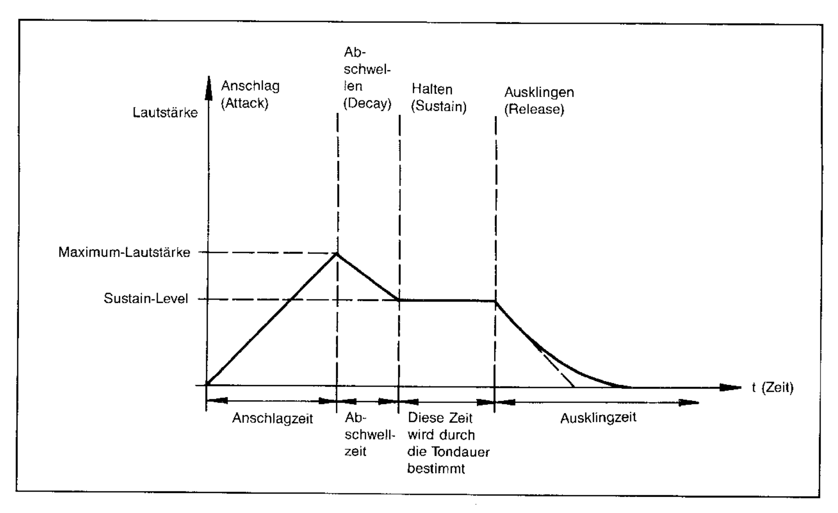


Bild 4/4.4-3

Wie aus Bild 4/4.4-3 ersichtlich, gibt der erste Parameter nach der Hüllkurvnummer (die Zeichnung stellt eine Hüllkurve dar) die Zeit an, die der Rechner benötigen soll, bis er von Lautstärke 0 bis zur eingestellten Lautstärke zum Anschwellen benötigen soll. Der nächste Parameter gibt die benötigte Zeit zur Verringerung der Lautstärke bis auf den Sustain-Level wieder. Der vierte Parameter in ENVELOPE-Befehl gibt jedoch nicht die Haltezeit an, sondern den sogenannten Sustain-Level, d.h. die Lautstärke, auf die der Ton nach dem Abschwellen abfällt. Die Haltezeit wird durch die verwendete Tondauer bestimmt. Der Parameter für die Ausklingzeit wird ausgewertet, wenn der Ton ausgeschaltet wurde, d.h. die Ausklingzeit ist nicht in der Tondauer enthalten. Dies ist besonders für Klangeffekte wichtig, da ein erneuter Ton das Ausklingen des vorhergehenden — innerhalb der gleichen Stimme — verhindert.

Damit haben Sie einen Überblick über die wichtigsten Parameter, die einen Ton beeinflussen können. Generell beeinflußt auch noch die Wellenform den Ton, wobei sich bei einer Dreieckwelle (Zeichnungen zu den Wellenformen siehe Handbuch Seite 4-156) ein leerer hohler Ton ergibt, die Sägezahn-Wellenform, die sicherlich nicht nur ihren Namen aufgrund der Wellenform hat, sondern sich auch so ähnlich anhört, bei einem Rechteckklang sich eine Mischung aus beiden vorgenannten ergibt, und ein Rauschen sicherlich nicht mehr als Ton anzusehen ist, was jedoch beim Schlagzeug nützliche Effekte gibt. Etwas abseits steht die Ringmodulation, wo die Töne teilweise etwas verzerrt klingen.

Die Hüllkurven (siehe Bild 4/4.4-3) stellen mit der gewählten Wellenform also die unterschiedlichen Klänge in den Tönen dar, wobei bei einer Rechteckwelle durch die Pulsbreite auch hier der Klang noch verändert werden kann. Damit Sie sehen, wie sich die Änderung der einzelnen Parameter auf die Töne auswirkt, wollen wir im folgenden ein kleines Programm vorstellen:

```
100 REM --- BEISPIEL ENVELOPE-BEFEHL ---
110 :
120 DIM WF$(4)
130 DATA DREIECK,SAEGEZAHN,RECHTECK,RAUSCHEN,RINGMODULATION
140 FOR I=0 TO 4
150 READ WF$(I)
160 NEXT
170 :
180 N=5
190 :
200 FOR AN=1 TO 15 STEP N
210 FOR AB=1 TO 15 STEP N
220 FOR HA=1 TO 15 STEP N
230 FOR AU=1 TO 15 STEP N
240 FOR WE=0 TO 4
250 :
260 PRINT"[ANSCHLAG      :        ";AN
270 PRINT"ABSCHWELLEN:        ";AB
280 PRINT"HALTEN       :        ";HA
290 PRINT"AUSKLINGEN  :        ";AU
300 PRINT"WELLENFORM  :        ";WF$(WE)
310 :
320 ENVELOPE 2,AN,AB,HA,AU,WE
330 PLAY"T2U6O3DCDEFGABO4C"
340 :
350 GETKEY A$
360 :
370 NEXT WE,AU,HA,AB,AN
380 :
```

Listing 4/4.4-4

Im Prinzip werden innerhalb von fünf ineinandergeschachtelten Schleifen jeweils nur die einzelnen Parameter beim ENVELOPE-Befehl umbesetzt. Um jedoch einen Hinweis über die aktuell „erklingenden“ Daten zu bekommen, werden diese zusätzlich am Bildschirm angezeigt.

Dazu wird zunächst in Zeile 120 ein Feld WF$() dimensioniert, indem die Namen der einzelnen Wellenformen untergebracht werden. Da sich das Anhören aller Möglichkeiten über einen etwas zu langen Zeitraum erstrecken würde (es gibt schließlich 327.680 Möglichkeiten, ohne Berücksichtigung der Impulsbreite bei Rechteckwellen), haben wir mit einem weiteren Parameter (N) eine Schrittweite vorgegeben, mit

der die einzelnen Parameter geändert werden sollen. Diese Schrittweite können Sie nach eigenen Wünschen ändern, jedoch sei aus erwähntem Grunde von der Schrittweite 1 abzuraten.

Ab Zeile 200 befinden sich die Schleifenbeginne der fünf Parameter, wobei die Variablen anhand der Angaben im Handbuch ausgewählt wurden. Ab Zeile 260 werden die aktuellen Daten ausgegeben, und in Zeile 320 wird der Hüllkurvennummer 2 der entsprechende Klang zugeordnet. Zeile 330 beinhaltet die Ausgabe einer einfachen Tonleiter. Wird eine Taste gedrückt, so wird die nächste Tonleiter erzeugt, wie es durch die Zeilen 350 und 370 vorgegeben ist.

Zur Ermittlung eines gewünschten Klanges können Sie auch einzelne Schleifen auslassen. Achten sie jedoch auf die Reihenfolge der Variablen beim NEXT-Befehl in Zeile 370. Bekanntermaßen verläuft diese Reihenfolge genau umgekehrt der Reihenfolge bei den FOR-Befehlen.

Damit haben Sie bereits die erste Möglichkeit, sich selbst eine Hüllkurve nach eigenen Wünschen zu definieren. Bis zu zehn solcher Hüllkurven können Sie definieren, da Sie alle vorgegebenen Hüllkurven nacheinander ersetzen können. Durch die Verwendung mehrerer ENVELOPE-Befehle innerhalb eines Programmes sind jedoch auch mehr Hüllkurven möglich (allerdings nicht gleichzeitig). Ebenso wie bei einer normalen Orgel können Sie mit Hilfe des ENVELOPE-Befehls also die Registrierung (Klangfarben) nach bestimmten Abschnitten in einem Musikstück ändern.

Vielleicht haben Sie aber auch schon bestimmte Vorstellungen über einen Klang und wollen nun noch etwas daran feilen. In diesem Fall hilft Ihnen sicherlich unser nächstes Programm.

Bedienungsanleitung

Nachdem Sie das Programm gestartet haben, können Sie mit den Ziffern 1 bis 8 alle Parameter des ENVELOPE-Befehls anwählen. Außerdem ist es möglich, eine gewünschte Melodie einzugeben. Nachdem Sie eine Ziffer eingegeben haben, erscheint am unteren Bildschirmrand die Frage „NEUER WERT?". Hier können Sie die neuen Daten eingeben. Bei der Wellenform genügt die Eingabe der der Wellenform voranstehenden Nummer. Die aktuelle Wellenform wird in inverser Schrift angezeigt. Wollen Sie den Ton/die Melodie mit den angezeigten Parametern spielen, so drücken Sie die Taste „9".

```
1000 REM ------------------------------------------------------------------------
1010 REM ---                        KLANGTEST                                ---
1020 REM ------------------------------------------------------------------------
1030 :
1040 REM ------------------------------------------------------------------------
1050 REM ---                  VARIABLEN VORBESETZEN                          ---
1060 REM ------------------------------------------------------------------------
1070 :
1080 REM ---                       UEBERSCHRIFT                              ---
1090 :
1100 UN$="_"
1110 OB$="‾"
1120 :
1130 FOR I=1 TO 7
1140 UN$=UN$+UN$
1150 OB$=OB$+OB$
1160 NEXT
1170 :
1180 UN$=LEFT$(UN$,80)
1190 OB$=LEFT$(OB$,80)
1200 :
1210 REM ---                       WELLENFORM                               ---
1220 :
1230 DIM WF$(4)
1240 :
1250 DATA 0-DREIECK,1-SAEGEZAHN,2-RECHTECK,3-RAUSCHEN,4-RINGMODULATION
1260 :
1270 FOR I=0 TO 4
1280 READ WF$(I)
1290 NEXT
1300 :
1310 REM ---           MELODIE UND 'ENVELOPE'-PARAMETER                     ---
1320 :
1330 PL$="O3CDEFGABO4C"
1340 :
1350 N=0
1360 AN=7
1370 AB=7
1380 HA=7
1390 AU=7
1400 IB=2048
1410 :
2000 REM -----------------------------------------------------------------------
2010 REM ---                  AUSGABE DER DATEN                             ---
2020 REM -----------------------------------------------------------------------
2030 :
2040 PRINT "□";UN$;
2050 PRINT"                          WAEHLEN SIE IHREN KLANG"
2060 PRINT OB$;
2070 :
2080 PRINT"▓▓1 - WELLENFORM: ";
2090 :
2100 FOR I=0 TO 4
2110 PRINT" ";
2120 IF WF=I THEN PRINT"▓";
2130 PRINT WF$(I);
```

Listing 4/4.4-5 (1)

```
2140 IF WF=I THEN PRINT"■";
2150 NEXT
2160 :
2170 PRINT
2180 :
2190 PRINT
2200 PRINT"2 - MELODIE          :    ■";PL$
2210 :
2220 PRINT
2230 PRINT"3 - NUMMER           :    ■";N
2240 PRINT"4 - ANSCHLAGZEIT     :    ■";AN
2250 PRINT"5 - ABSCHWELLZEIT    :    ■";AB
2260 PRINT"6 - HALTEZEIT        :    ■";HA
2270 PRINT"7 - AUSKLINGZEIT     :    ■";AU
2280 PRINT
2290 :
2300 PRINT"8 - IMPULSBREITE     :    ■";IB
2310 :
2320 PRINT"■■■■■WELCHEN WERT WOLLEN SIE AENDERN (1-8) ? (SPIELEN=9)"
2330 GETKEY A$
2340 A=VAL(A$)
2350 IF A<1 OR A>9 THEN 2330
2360 :
2370 ON A GOTO 3000,3100,3200,3300,3400,3500,3600,3700,3800
2380 :
3000 REM --- WELLENFORM AENDERN ---
3010 :
3020 INPUT"■■NEUER WERT";WF
3030 IF WF<0 OR WF>4 THEN 3020
3040 ENVELOPE N,AN,AB,HA,AU,WF,IB
3050 GOTO 2040
3060 :
3100 REM --- MELODIE AENDERN ---
3110 :
3120 INPUT"■■NEUER WERT";PL$
3130 ENVELOPE N,AN,AB,HA,AU,WF,IB
3140 GOTO 2040
3150 :
3200 REM --- NUMMER AENDERN ---
3210 :
3220 INPUT"■■NEUER WERT";N
3230 ENVELOPE N,AN,AB,HA,AU,WF,IB
3240 GOTO 2040
3250 :
3300 REM --- ANSCHLAGZEIT AENDERN ---
3310 :
3320 INPUT"■■NEUER WERT";AN
3330 ENVELOPE N,AN,AB,HA,AU,WF,IB
3340 GOTO 2040
3350 :
3400 REM --- ABSCHWELLZEIT AENDERN ---
3410 :
3420 INPUT"■■NEUER WERT";AB
3430 ENVELOPE N,AN,AB,HA,AU,WF,IB
3440 GOTO 2040
3450 :
```

Listing 4/4.4-5 (2)

```
3500 REM --- HALTEZEIT AENDERN ---
3510 :
3520 INPUT"██NEUER WERT";HA
3530 ENVELOPE N,AN,AB,HA,AU,WF,IB
3540 GOTO 2040
3550 :
3600 REM --- AUSKLINGZEIT AENDERN ---
3610 :
3620 INPUT"██NEUER WERT";AU
3630 ENVELOPE N,AN,AB,HA,AU,WF,IB
3640 GOTO 2040
3650 :
3700 REM --- IMPULSBREITE AENDERN ---
3710 :
3720 INPUT"██NEUER WERT";IB
3730 ENVELOPE N,AN,AB,HA,AU,WF,IB
3740 GOTO 2040
3750 :
3800 REM --- MELODIE SPIELEN ---
3810 :
3830 PLAY "T"+RIGHT$(STR$(N),1)+PL$
3840 GOTO 2330
3850 END
```

Listing 4/4.4-5 (3)

Programmbeschreibung

Sicherlich genügt das dargestellte Programm nur minimal den Anforderungen. Wer sich schon ein bißchen in der Grafik auskennt, kann sich vielleicht die durch die Parameter gegebene Hüllkurve zusätzlich auf dem Bildschirm darstellen. Auch die Ansteuerung und Änderung der einzelnen Parameter über Joystick dürfte bei entprechenden Vorkenntnissen leicht zu machen sein.

Beginnen wir jedoch mit der Besprechung des vorliegenden Programmes. In den beiden Parametern UN$ und OB$ werden die Über- und Unterstreichungsstriche für die Überschrift in einem einzigen Zeichen festgehalten. Die Grafikzeichen finden Sie auf der Tastatur. Im weiteren lernen Sie einen kleinen Trick kennen, wie man Zeichenreihen aus den gleichen Zeichen mit einer bestimmten Länge definieren kann, ohne diese bei der Eingabe abzuzählen. Zuerst werden dazu die Zeichenreihen entsprechend oft verdoppelt, in unserem Fall siebenmal. Dann wird mit Hilfe des LEFT$()-Befehls die gewünschte Anzahl von Zeichen abgeschnitten.

Wie in den bisher vorgestellten Programmen auch, wird im weiteren zunächst ein Feld (WF$) definiert, in dem die Texte zur Bezeichnung der verschiedenen Wellenform abgelegt werden. Als Vorgabe für den PLAY-Befehl dient uns die Variable PL$, in der wir eine Tonleiter in der dritten Oktave abgelegt haben. Für unsere Zwecke wird zunächst die Hüllkurve Nummer 0 ausgewählt, die Sie jedoch später ändern können. Wenn Sie also einen gewünschten Klang gefunden haben und weiter experimentieren wollen, so ändern Sie — nachdem Sie sich die Daten vielleicht notiert haben — die Hüllkurvennummer, und die fertige Klangfarbe bleibt erhalten. Für Anschlagzeit, Abschwellzeit, Haltelevel und Ausklingzeit wurden jeweils mittlere Werte vorgegeben. Ebenfalls für die Impulsbreite, die jedoch nur beim Rechteckklang wichtig ist.

Nachdem in Zeile 40 zunächst der Bildschirm gelöscht wurde, wird der Unterstreichungsstrich die Überschrift und der Oberstreichungsstrich ausgegeben. Anschließend erfolgt die Ausgabe der Wellenform, wobei jedoch — durch die Variable WF gekennzeichnet — die aktuelle Wellenform in reverser Schrift dargestellt wird. Dies wird erreicht, indem die reverse Schrift in Zeile 2120 bei der entsprechenden Wellenform eingeschaltet wird, und in Zeile 2140 dieser Zustand rückgängig gemacht wird.

Dann werden ab Zeile 2190 die weiteren Parameter am Bildschirm ausgegeben, ebenso die Frage nach dem Änderungswunsch an den Anwender. Es folgt die übliche Warteschleife mit der obligatorischen Plausibilitätsprüfung und anschließendem Programmverteiler auf die entsprechende Zeilengruppe ab Zeile 3000.

Die Unterprogramme zum Ändern der einzelnen Parameter sind alle in der gleichen Form aufgebaut. Zunächst erfolgt die Frage „NEUER WERT“ an den Anwender, womit der neue Wert der entsprechenden Variablen (WF: Wellform, PL$: Melodie; N: Hüllkurvennummer; AN: Anschlagzeit; AB: Abschwellzeit; HA: Haltezeit; AU: Ausklingzeit und IB: Impulsbreite) erfaßt wird. Um ungewünschte Reaktionen zu vermeiden, wird bei der Wellenform noch eine Plausibilitätsprüfung durchgeführt. Bei den anderen Parametern ist dies relativ unnötig, da jeweils nur ein Wert modulo 15 (bzw. ein Text bei einer Melodie) angenommen wird. Auf Wunsch können Sie auch hier noch Plausibilitätsprüfungen einfügen.

Der Übersichtlichkeit halber sind beim ENVELOPE-Befehl jeweils alle Parameter aufgeführt, obwohl die nicht geänderten — wie eingangs des Kapitels erwähnt — weggelassen werden könnten.

Zum Spielen der Melodie wird zunächst hinter dem Buchstaben „T“ die um das Leerzeichen (positive Vorzeichen) gekürzte, in eine Zeichenreihe umgewandelte Zahl an den Rechner im PLAY-Befehl übergeben, anschließend die Melodie.

Mit den beiden in diesem Kapitel vorgestellten Programmen haben Sie bereits die Möglichkeit, sich einen Klang nach Ihren Wünschen auszusuchen, mit dem Sie beim PLAY-Befehl arbeiten können.

Sobald Sie die Daten eines gewünschten Klanges herausgefunden haben, sollten Sie sich diese Daten auf jeden Fall notieren, da nach erneutem Einschalten des Rechners die Klangfarben wieder die vorgegebenen Werte — siehe Handbuch — beinhalten.

Aber mit dem ENVELOPE-Befehl sind noch nicht alle Möglichkeiten ausgeschöpft, den Klang zu beeinflussen. Eine weitere Möglichkeit bildet die FILTER-Anweisung, mit der Sie Feinheiten des Klanges einstellen können.

4/4.4.3

Filter — den Klang verfeinern

Wie bereits erwähnt, bildet der FILTER-Befehl eine weitere Möglichkeit, den Klang — wenn auch nur in geringem Umfange — zu ändern. Über Filter kann man sehrviel und ausführlich berichten, was jedoch den Rahmen dieses Buches sprengen würde. Die Möglichkeit der Filter sind im Handbuch des C 128 ab Seite 4-162 sehr gut beschrieben, so daß wir uns hier auf den Einbau der Filteranweisung in das zuletzt dargestellte Programm beschränken wollen. Auf jeden Fall sollten Sie die Möglichkeit des Filters ausnutzen, aber vorher ausgiebig mit diesem Befehl herumprobieren.

Da wir auch einige wenige Zeilen innerhalb des bereits aufgeführten Programmes geändert haben, wollen wir im folgenden das komplette Listing ausdrucken, was auch denen zugute kommt, die den ENVELOPE-Befehl in diesem Buch übersprungen haben.

```
1000 REM --------------------------------------------------------------------
1010 REM ---                          KLANGTEST                          ---
1020 REM --------------------------------------------------------------------
1030 :
1040 REM --------------------------------------------------------------------
1050 REM ---                   VARIABLEN VORBESETZEN                     ---
1060 REM --------------------------------------------------------------------
1070 :
1080 REM ---                         UEBERSCHRIFT                        ---
1090 :
1100 UN$="_"
1110 OB$="¯"
1120 :
1130 FOR I=1 TO 7
1140 UN$=UN$+UN$
1150 OB$=OB$+OB$
1160 NEXT
1170 :
1180 UN$=LEFT$(UN$,80)
1190 OB$=LEFT$(OB$,80)
1200 :
1210 REM ---                          WELLENFORM                         ---
1220 :
1230 DIM WF$(4)
1240 :
1250 DATA 0-DREIECK,1-SAEGEZAHN,2-RECHTECK,3-RAUSCHEN,4-RINGMODULATION
1260 :
1270 FOR I=0 TO 4
1280 READ WF$(I)
1290 NEXT
1300 :
1310 REM ---            MELODIE UND 'ENVELOPE'-PARAMETER                 ---
1320 :
1330 PL$="O3CDEFGABO4C"
1340 :
1350 N=0
1360 AN=7
1370 AB=7
1380 HA=7
1390 AU=7
1400 IB=2048
1410 :
2000 REM --------------------------------------------------------------------
2010 REM ---                     AUSGABE DER DATEN                       ---
2020 REM --------------------------------------------------------------------
2030 :
2040 PRINT "[clr]";UN$;
2050 PRINT"                              WAEHLEN SIE IHREN KLANG"
2060 PRINT OB$;
2070 :
2080 PRINT"[ctrl]1 - WELLENFORM: ";
2090 :
2100 FOR I=0 TO 4
2110 PRINT" ";
2120 IF WF=I THEN PRINT"[rvs]";
2130 PRINT WF$(I);
```

Listing 4/4.4-6 (1)

```
2140 IF WF=I THEN PRINT"■";
2150 NEXT
2160 :
2170 PRINT
2180 :
2190 PRINT
2200 PRINT"2 - MELODIE           :    ▮";PL$
2210 :
2220 PRINT
2230 PRINT"3 - NUMMER            :    ▮";N
2240 PRINT"4 - ANSCHLAGZEIT      :    ▮";AN
2250 PRINT"5 - ABSCHWELLZEIT     :    ▮";AB
2260 PRINT"6 - HALTEZEIT         :    ▮";HA
2270 PRINT"7 - AUSKLINGZEIT      :    ▮";AU
2280 PRINT
2290 :
2300 PRINT"8 - IMPULSBREITE      :    ▮";IB
2310 :
2315 GOSUB 5000
2320 PRINT"▮▮WELCHEN WERT WOLLEN SIE AENDERN (1-8) ? (SPIELEN=9 / FILTER=F)"
2330 GETKEY A$
2335 IF A$="F" THEN 4000
2340 A=VAL(A$)
2350 IF A<1 OR A>9 THEN 2330
2360 :
2370 ON A GOTO 3000,3100,3200,3300,3400,3500,3600,3700,3800
2380 :
3000 REM --- WELLENFORM AENDERN ---
3010 :
3020 INPUT"▮▮NEUER WERT";WF
3030 IF WF<0 OR WF>4 THEN 3020
3040 ENVELOPE N,AN,AB,HA,AU,WF,IB
3050 GOTO 2040
3060 :
3100 REM --- MELODIE AENDERN ---
3110 :
3120 INPUT"▮▮NEUER WERT";PL$
3130 ENVELOPE N,AN,AB,HA,AU,WF,IB
3140 GOTO 2040
3150 :
3200 REM --- NUMMER AENDERN ---
3210 :
3220 INPUT"▮▮NEUER WERT";N
3230 ENVELOPE N,AN,AB,HA,AU,WF,IB
3240 GOTO 2040
3250 :
3300 REM --- ANSCHLAGZEIT AENDERN ---
3310 :
3320 INPUT"▮▮NEUER WERT";AN
3330 ENVELOPE N,AN,AB,HA,AU,WF,IB
3340 GOTO 2040
3350 :
3400 REM --- ABSCHWELLZEIT AENDERN ---
3410 :
3420 INPUT"▮▮NEUER WERT";AB
3430 ENVELOPE N,AN,AB,HA,AU,WF,IB
3440 GOTO 2040
3450 :
```

Listing 4/4.4-6 (2)

```
3500 REM --- HALTEZEIT AENDERN ---
3510 :
3520 INPUT"NEUER WERT";HA
3530 ENVELOPE N,AN,AB,HA,AU,WF,IB
3540 GOTO 2040
3550 :
3600 REM --- AUSKLINGZEIT AENDERN ---
3610 :
3620 INPUT"NEUER WERT";AU
3630 ENVELOPE N,AN,AB,HA,AU,WF,IB
3640 GOTO 2040
3650 :
3700 REM --- IMPULSBREITE AENDERN ---
3710 :
3720 INPUT"NEUER WERT";IB
3730 ENVELOPE N,AN,AB,HA,AU,WF,IB
3740 GOTO 2040
3750 :
3800 REM --- MELODIE SPIELEN ---
3810 :
3830 PLAY "X1T"+RIGHT$(STR$(N),1)+PL$
3840 GOTO 2330
3850 END
3860 :
3870 SCRATCH"ENVELOPE2":PRINTDS$:DSAVE"ENVELOPE2":PRINTDS$:END
4000 REM ------------------------------------------------------------
4010 REM ---                  FILTERDATEN AENDERN                ---
4020 REM ------------------------------------------------------------
4030 :
4040 INPUT"NEUE FREQUENZ";FF
4050 PRINT"                                          "
4060 INPUT"TIEFPASS (EIN=1/AUS=0)";TP
4070 PRINT"                                          "
4080 INPUT"BANDPASS (EIN=1/AUS=0)";BP
4090 PRINT"                                          "
4100 INPUT"HOCHPASS (EIN=1/AUS=0)";HP
4110 PRINT"                                          "
4120 INPUT"NEUE RESONANZ";RE
4130 :
4140 FILTER FF,TP,BP,HP,RE
4150 :
4160 GOTO 2040
4170 :
5000 REM ------------------------------------------------------------
5010 REM ---        UNTERPROGRAMM: FILTERDATEN ANZEIGEN          ---
5020 REM ------------------------------------------------------------
5030 :
5040 PRINT
5050 PRINT"F - FILTER:     FREQUENZ ";FF;"      ";
5060 IF TP THEN PRINT"TIEF      ";
5070 IF TP=0 THEN PRINT"TIEF      ";
5080 IF BP THEN PRINT"BAND      ";
5090 IF BP=0 THEN PRINT"BAND      ";
5100 IF HP THEN PRINT"HOCH      "
5110 IF HP=0 THEN PRINT"HOCH      "
5120 PRINT"                 RESONANZ ";RE
5130 :
5140 RETURN
```

Listing 4/4.4-6 (3)

Die hier nicht beschriebenen Zeilen sind zum Schluß des vorangegangenen Kapitels erklärt. Innerhalb des Listings wurden nur einige wenige Zeilen geändert, so wurde z.B. die Zeile 2315 eingefügt, wo das Unterprogramm zur Ausgabe der Filterdaten auf dem Bildschirm aufgerufen wird. Außerdem wurde Zeile 2335 eingefügt, die als Eingabetaste auch noch ein „F“ zuläßt, wonach zu Zeile 4000 verzweigt wurde. Außerdem wurde die Anzahl der „Cursor nach unten“ (Reverse Q) in Zeile 2320 um drei Stück vermindert. Achten Sie unbedingt darauf, daß die Abfrage nach dem „F“ vor der Plausibilitätsprüfung erfolgt, da sonst der Programmteil nicht angesteuert würde.

Für die Filterwerte wurden folgende Variablen verwendet:

FF :	Filterfrequenz	
TP :	Merker für Tiefpass	(1=ein; 0=aus)
BP :	Merker für Bandpass	(1=ein; 0=aus)
HP :	Merker für Hochpass	(1=ein; 0=aus)
RE :	Resonanz (Werte 0-15)	

Diese Werte werden ab Zeile 4040 erfaßt. Da hier mehrere Werte gleichzeitig zu erfassen sind, haben wir uns eines kleinen Trickes beholfen. Die Cursorsteuerzeichen bei den PRINT- und INPUT-Befehlen bedeuten jeweils einmal „Cursor nach oben“, so daß nach der Erfassung eines Wertes jeweils diese Zeile durch die nächste Anweisung gelöscht wird. In die gleiche Zeile wird durch das erwähnte „Cursor nach oben“ der Text für den nächsten INPUT-Befehl gesetzt.

Sofern Sie nicht alle Werte ändern wollen, hilft Ihnen eine Eigenschaft des C 128: Wird bei einem INPUT-Befehl keine Eingabe gemacht, so wird der Wert der Variablen nicht geändert. In Zeile 4140 werden die Daten ausgegeben, und anschließend wird zur Ausgabe der gewählten Daten auf dem Bildschirm übergegangen.

Ab Zeile 5000 haben wir ein Unterprogramm zur Ausgabe der Filterdaten angehangen, da hierfür im Rahmen des gegebenen Programmes zu wenig Platz gewesen wäre, und die Zeilennummern des Basic-Programmes nicht geändert werden sollten. Auch hier werden wieder die angewählten Filterarten (Tiefpass, Bandpass, Hochpass) durch inverse Schrift dargestellt, sofern sie angewählt sind. Die Filterfrequenz und der Resonanzwert wird jeweils zahlenmäßig ausgegeben.

Alle Theorie ist grau! Probieren geht über Studieren!

4/4.4.4

TEMPO — wählen Sie das richtige Tempo

Der TEMPO-Befehl soll hier nur der Vollständigkeit halber erwähnt werden. Auch in der Musik ist eine Viertelnote nicht gleich eine Viertelnote — auch wenn sie die gleiche Notendarstellung haben. Diese unterschiedlich dauernden Musiknoten können Sie mit dem TEMPO-Befehl beeinflussen.

Aber auch noch andere Einsatzmöglichkeiten gibt es für den TEMPO-Befehl. Wenn Sie z.B. in unserem ersten Beispielprogramm beim ENVELOPE-Befehl verschiedene Tempo-Ziffern eingeben, so können Sie die Auswirkungen leicht selbst feststellen. Schnell aufeinanderfolgende Töne (TEMPO über 200) in Verbindung mit Rauschen und bei Verwendung einer einfachen Tonleiter mit langer Ausklingdauer (über 10) kann man sehr leicht zur akustischen Darstellung eines Maschinengewehrs verwenden.

Auch hier hilft das Herumprobieren, und mit dem bisher Aufgezeigten sollte es ein leichtes für Sie sein, auch noch eine Änderung des Tempos in die vorgestellten Programme einzubauen.

4/4.4.5

SOUND — Geräusche nach Wahl

Bisher haben wir uns nur mit „normalen“ Tönen beschäftigt. Mit dem SOUND-Befehl haben Sie jedoch die Möglichkeit, beliebige Klangeffekte zu erzeugen.

War der ENVELOPE-Befehl ein Hilfsmittel, um Klangfarben für mit dem PLAY-Befehl gespielten Notenwerte zu ändern, so bietet der SOUND-Befehl beides in einem. Auch hier sollten Sie sich durch die vielen Klammern im Handbuch beim Format nicht beeindrucken lassen. Diese sagen lediglich aus, daß die Stimme, die Frequenz und die Dauer eingegeben werden müssen, Richtung, Maximalfrequenz,

Stufe, Wellenform und Impulsbreite jedoch optional sind. Auch hier gilt wieder, daß fehlende Parameter möglich sind, die zwischen den Parametern stehenden Kommata jedoch zu setzen sind.

Ein Beispielprogramm ähnlich unserem ersten beim ENVELOPE-Befehl ist an dieser Stelle nicht sehr sinnvoll, auch wenn man für Frequenz, Dauer, Maximalfrequenz und Stufe in den FOR . . .NEXT-Schleifen jeweils große Schrittweiten heranzieht. Auf jeden Fall empfiehlt sich aber die Umarbeitung des zweiten Beispielprogrammes. Sie erinnern sich: Alle Parameter des ENVELOPE-Befehls konnten durch Eingabe am Bildschirm geändert werden. Dieses kleine Projekt wollen wir im folgenden in Angriff nehmen:

```
1000 REM ---------------------------------------------------------------------
1010 REM ---                          SOUNDTEST                           ---
1020 REM ---------------------------------------------------------------------
1030 :
1040 REM ---------------------------------------------------------------------
1050 REM ---                    VARIABLEN VORBESETZEN                     ---
1060 REM ---------------------------------------------------------------------
1070 :
1080 REM ---                         UEBERSCHRIFT                         ---
1090 :
1100 UN$="_"
1110 OB$="¯"
1120 :
1130 FOR I=1 TO 7
1140 UN$=UN$+UN$
1150 OB$=OB$+OB$
1160 NEXT
1170 :
1180 UN$=LEFT$(UN$,80)
1190 OB$=LEFT$(OB$,80)
1200 :
1210 REM ---                         WELLENFORM                          ---
1220 :
1230 DIM WF$(3)
1240 :
1250 DATA 0-DREIECK,1-SAEGEZAHN,2-RECHTECK,3-RAUSCHEN
1260 :
1270 FOR I=0 TO 3
1280 READ WF$(I)
1290 NEXT
1300 :
1310 REM ---                       SOUND-PARAMETER                        ---
1320 :
1330 SI=1
1340 FQ=2000
1350 DA=180
1360 RI=0
1370 MA=5000
1380 SU=3000
1390 WF=3
1400 IB=2000
1410 :
```

Listing 4/4.4-7 (1)

```
2000 REM -----------------------------------------------------------------
2010 REM ---                   AUSGABE DER DATEN                       ---
2020 REM -----------------------------------------------------------------
2030 :
2040 PRINT "♥";UN$;
2050 PRINT"                           WAEHLEN SIE IHREN SOUND"
2060 PRINT OB$;
2070 :
2080 PRINT"██1 - WELLENFORM:  ";
2090 :
2100 FOR I=0 TO 3
2110 PRINT" ";
2120 IF WF=I THEN PRINT"█";
2130 PRINT WF$(I);
2140 IF WF=I THEN PRINT"█";
2150 NEXT
2160 :
2170 PRINT
2180 :
2190 PRINT
2200 PRINT"2 - FREQUENZ         :    █";FQ
2210 PRINT
2220 PRINT"3 - STIMME           :    █";SI
2230 PRINT"4 - DAUER            :    █";DA
2240 PRINT"5 - RICHTUNG         :    █";RI
2250 PRINT"6 - MAXIMALFREQUENZ:      █";MA
2260 PRINT"7 - STUFE            :    █";SU
2270 PRINT
2280 :
2290 PRINT"8 - IMPULSBREITE     :    █";IB
2300 :
2310 PRINT"█████WELCHEN WERT WOLLEN SIE AENDERN (1-8 / 9=NUR SOUND) ?"
2320 GETKEY A$
2330 A=VAL(A$)
2340 IF A<1 OR A>9 THEN 2320
2350 :
2360 ON A GOTO 3000,3100,3200,3300,3400,3500,3600,3700,3800
2370 :
3000 REM --- WELLENFORM AENDERN ---
3010 :
3020 INPUT"██NEUER WERT";WF
3030 IF WF<0 OR WF>3 THEN 3020
3040 SOUND SI,FQ,DA,RI,MA,SU,WF,IB
3050 GOTO 2040
3060 :
3100 REM --- FREQUENZ AENDERN ---
3110 :
3120 INPUT"██NEUER WERT";FQ
3130 SOUND SI,FQ,DA,RI,MA,SU,WF,IB
3140 GOTO 2040
3150 :
3200 REM --- STIMME AENDERN ---
3210 :
3220 INPUT"██NEUER WERT";SI
3230 SOUND SI,FQ,DA,RI,MA,SU,WF,IB
3240 GOTO 2040
3250 :
```

Listing 4/4.4-7 (2)

```
3300 REM --- DAUER AENDERN ---
3310 :
3320 INPUT"██NEUER WERT";DA
3330 SOUND SI,FQ,DA,RI,MA,SU,WF,IB
3340 GOTO 2040
3350 :
3400 REM --- RICHTUNG AENDERN ---
3410 :
3420 INPUT"██NEUER WERT";RI
3430 SOUND SI,FQ,DA,RI,MA,SU,WF,IB
3440 GOTO 2040
3450 :
3500 REM --- MAXFREQUENZ AENDERN ---
3510 :
3520 INPUT"██NEUER WERT";MA
3530 SOUND SI,FQ,DA,RI,MA,SU,WF,IB
3540 GOTO 2040
3550 :
3600 REM --- STUFE AENDERN ---
3610 :
3620 INPUT"██NEUER WERT";SU
3630 SOUND SI,FQ,DA,RI,MA,SU,WF,IB
3640 GOTO 2040
3650 :
3700 REM --- IMPULSBREITE AENDERN ---
3710 :
3720 INPUT"██NEUER WERT";IB
3730 SOUND SI,FQ,DA,RI,MA,SU,WF,IB
3740 GOTO 2040
3750 :
3800 REM --- NUR SOUND ---
3810 :
3820 SOUND SI,FQ,DA,RI,MA,SU,WF,IB
3830 GOTO 2320
```

Listing 4/4.4-7 (3)

Mit diesem Listing werden Sie wohl kaum noch Schwierigkeiten haben, sowohl das Zirpen einer Grille, aber auch das Geheul einer amerikanischen Polizeisirene nachzumachen. Ebenso das Verschwinden eines Raumschiffes im tiefen Raum oder akustische Klangeffekte für Spiele bereiten keine Schwierigkeiten mehr, sofern Sie sich durch einiges Probieren die Bedeutung der einzelnen Parameter klargemacht haben.

Hier jedoch zunächst die Programmbeschreibung, die wir nicht mehr so ausführlich halten wollen, da wir beim ENVELOPE-Befehl ein ähnliches Programm bereits vorgestellt haben.

Auch bei diesem Listing beginnen wir wieder mit den Variablen für die Überschrift (UN$;OB$). Bei der Wellenform ist zu berücksichtigen, daß eine Ringmodulation beim SOUND-Befehl nicht mehr zur Verfügung steht, demgemäß nur noch vier Wellenformen auszuwählen sind und WF$() nur mit „3“ zu dimensionieren ist.

Für den Sound wurden grundsätzlich andere Variablen herangezogen, so daß eine mnemotechnische Bezeichnung wiederum gewährleistet ist. Die Variablen haben folgende Bedeutung:

SI	: Stimme (Die Variable ST ist für den Status reserviert)
FQ	: Freqenz
DA	: Dauer
RI	: Richtung
MA	: Maximalfrequenz
SU	: Stufe
WF	: Wellenform
IB	: Impulsbreite

Noch etwas zur Maximalfrequenz: Um die Richtung (auch ein sehr wichtiger Faktor) voll zur Geltung zu bringen, sollte auf jeden Fall die Maximalfrequenz niedriger als die eingestellte Frequenz (FQ) sein, da sich sonst keine an- und abschwellenden Effekte ergeben.

Die Bezeichnung im Handbuch ist zwar etwas mißverständlich, trotzdem haben wir sie beibehalten. Im Prinzip funktioniert das An- und Abschwellen zwischen den beiden Frequenzen, die wir in den Variablen FQ und MA festhalten.

Wie bei unserem ENVELOPE-Programm wird zunächst die Wellenform und — als wichtigstes, abgesetzt von den anderen Parametern — die Frequenz angegeben. Es folgen die weiteren Parameter und die Abfrage für den Anwender. Im folgenden wurde die Ausgabe des Tones so gehandhabt, daß der SOUND nach jeder Änderung ertönt, mit einem Druck auf die Taste „G“ läßt sich jedoch das Geräusch auch ohne Änderung erfassen.

Das Erfassen und die Umbesetzung der Parameter beim SOUND-Befehl erfolgt analog dem bereits beschriebenen Programm.

Sofern Sie das eine Programm aus dem anderen Programm durch Ändern erreichen wollen, achten Sie darauf, daß Sie jeweils den ENVELOPE-Befehl durch den SOUND-Befehl (richtige Parameter!) ersetzen.

Melodien mit dem SOUND-Befehl zu spielen, sollten Sie vermeiden, dafür ist er nicht gedacht. Was sind nun die besonderen Eigenschaften des SOUND-Befehls? Um die Wirkung des SOUND-Befehls zu beschreiben, betrachten wir uns Bild 4/4.4-4.

Wenn man den Einfluß auf die Parameter erst einmal kennt, ist es nicht weiter schwierig, ein gewünschtes Geräusch durch kurzes Probieren herauszufinden.

Dargestellt wurde ein anschwellender Ton, der durch den Richtungsparameter 0 erzeugt wird. Würde der Richtungsparameter „1“ betragen, so würden die Kurven nicht von unten nach oben, sondern von oben nach unten verlaufen. Wäre der Richtungsparameter hingegen „2“, so würde keine senkrechte Flanke entstehen, sondern der Ton ebenso abschwellen wie er angeschwellt ist. Der Bereich, in dem der Ton hin und her schwellt, ist durch die beiden Parameter Frequenz und Maximalfrequenz (besser Minimalfrequenz) gegeben. Die Dauer bestimmt sinngemäß die Dauer des Tones, und die Schnelligkeit des An- und Abschwellens wird durch den Parameter Stufe dargestellt. Die Stufen sind in der Zeichnung auch angedeutet.

Im Prinzip sehen Sie nichts anderes als eine Art von Sägezahnkurve. Beachten Sie jedoch dabei, daß sich innerhalb der Kurve noch eine Dreieck-, Sägezahn- oder Rechteckschwingung befinden kann bzw. ein oszillierendes Rauschen.

Auf jeden Fall sollten die Parameter sehr gut aufeinander abgestimmt sein. In unserem Beispiel wird der Ton beim zweiten Abschwellen abgebrochen, da der Zeitraum, der durch den Parameter Dauer bestimmt ist, zu Ende war. Eine zu kurze Dauer, wie sie ebenfalls in unserem Beispiel dargestellt ist, würde z.B. dafür sorgen, daß der Ton noch nicht mal bis zu seiner höchsten Frequenz anschwellen kann.

Nachdem Sie nun bereits etwas über den Einfluß der Parameter beim SOUND-Befehl wissen, wollen wir im folgenden ein paar Beispiele durchspielen, die das zeichnerisch Dargestellte weiter verdeutlichen.

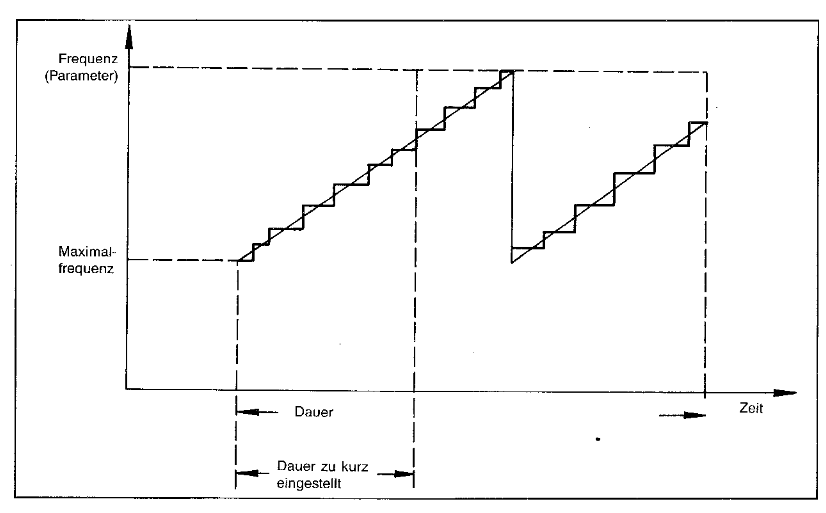


Bild 4/4.4-4 Parameter beim SOUND-Befehl (Richtung = 0: zunehmende Klangstufe)

Stellen Sie zunächst die Parameter wie folgt ein:

Wellenform	: Rechteck
Frequenz	: 5000
Stimme	: 1
Dauer	: 300
Richtung	: 0
Maximalfrequenz	: 2000
Stufe	: 50
Impulsbreite	: 2000

Wenn Sie nun die Taste „G“ drücken, ertönt ein öfters anschwellender Ton, wie ihn vielleicht eine Signalsirene hervorrufen würde. Als erstes wollen wir den Parameter Nummer 7 (Stufe) ändern. Zunächst setzen wir ihn auf den Wert 100. Unser Heulen geht nun doppelt so schnell vonstatten wie vorher. Ist der Ton vorher fünfmal von unten nach oben in der Tonhöhe angeschwollen, so schwillt er jetzt zehnmal an. Den gegenteiligen Effekt erreichen Sie, wenn Sie bei Stufe den Wert 10 eingeben: Jetzt schwillt der Ton nur einmal an.

Wenn Sie bei Stufe den Wert 500 eingeben, ergibt sich ein Sound, wie Sie ihn vielleicht von Spielautomaten aus Spielhallen kennen. Etwas schneller wird er natürlich noch beim Wert von 1000, und bei einem Wert von 32500 ist kein Unterschied zu erkennen. Die Grenze, wo man noch unterschiedliche Töne (ein Oszillieren) hören kann, liegt etwa bei einer Stufenhöhe von 3000. Die Stufe ist also für die Steilheit (der aufsteigenden Geraden in unserer Zeichnung) des an- bzw. abschwellenden Tones zuständig und bestimmt u.a. zusammen mit der Dauer, wie oft ein Ton an- und abschwillt. Kleine Werte bedeuten nach unseren Versuchen also ein langsames An- und Abschwellen, große Werte ein — meist nicht mehr hörbares — Vibrieren.

Setzen wir nun unseren Stufenwert wieder auf 100 und verändern die Richtung, indem wir ihr den Wert „1“ geben. Wir hören nun keine anschwellenden Töne mehr, sondern abschwellende, was sofort einen neuen Geräuscheffekt ergibt.

Als nächstes ändern wir den Richtungsparameter noch auf den Wert „2“, was einen an- und abschwellenden Ton erzeugt. Wenn Sie nun die Stufe wieder auf den Wert 50 bringen, kommt Ihnen das Geräusch sicherlich bekannt vor.

Nun probieren Sie die verschiedenen Wellenformen, indem Sie nacheinander die Parameter 0, 1 und 3 ebenfalls ausprobieren. Für das Heulen einer (amerikanischen) Polizeisirene ist sicherlich die Sägezahn-Wellenform besser geeignet. Bei der Wellenform Rauschen sollte man vielleicht noch etwas mit der Geschwindigkeit des An- und Abschwellens heruntergehen und die Stufenzahl eventuell auf den Wert 25 setzen, was Ähnlichkeit mit dem Geräusch eines brausenden Sturmes hat.

In diesem Fall kann man die obere Frequenzgrenze auch etwas höher ansetzen und die Stufenzahl wieder auf 100 erhöhen. Wenn Sie nun den Richtungsparameter auf den Wert „1“ setzen, die Stufe auf 200 und die Frequenz auf 2000, befinden Sie sich mitten in einer Weltraumschlacht. Diesen Effekt können Sie erhöhen, wenn Sie die Dauer auf 600 hochsetzen und anschließend durch schnelles Hintereinanderdrücken der Tasten 3, 1, RETURN, 3, 2, RETURN, 3, 3, RETURN zu einem wahren Weltraum-Inferno werden lassen. Sie haben das Geräusch nun auf allen drei Stimmen, und durch die fortlaufende Eingabe auch zeitlich versetzt.

Wenn Sie jetzt auf Rechteck-Wellenform umschalten, haben Sie auch gleich den Gegner.

Geben Sie folgende Parameter ein:

Wellenform	: Rechteck
Frequenz	: 60000
Stimme	: 1
Dauer	: 300
Richtung	: 1
Maximalfrequenz	: 58000
Stufe	: 300
Impulsbreite	: 500

Erkennen Sie das Geräusch?

Und nun viel Spaß beim Probieren.

4/4.4.6

VOL — wie laut soll's denn sein?

Der VOL-Befehl soll hier nur der Vollständigkeit halber erwähnt werden. Da die Befehle PLAY, ENVELOPE, FILTER und TEMPO in gewisser Weise zusammen gehören, beim PLAY-Befehl hinter einem U auch die Lautstärke angegeben werden kann, erübrigt sich hier ein VOL-Befehl weitestgehend. Sinnvoll ist der VOL-Befehl jedoch in Zusammenhang mit SOUND.

4/4.6

Mausprogrammierung

Autor: Rainer König

Die Maus als Hilfsmittel zur interaktiven Benutzerführung ist heute in aller Munde. Erstmals trat sie bei Apple's LISA auf, ihr folgten die optische Maus von TELE-VIDEO und die Maus von Microsoft (hier in Verbindung mit dem Textprocessor ‚WORD' der Firma Microsoft). Auch Apple's MacIntosh erhielt wieder die bereits von LISA her bekannte Maus. Was sind nun aber die Eigenschaften einer Computer-Maus?

Eine Maus ist für den Benutzer nichts anderes als ein Schiebekasten (in der Fachwelt heißt das umgedrehte Rollkugel) mit einem oder mehreren Tasten. Indem der Benutzer dieses Kästchen auf dem Schreibtisch hin- und herschiebt, kann er einen Pfeil oder ein ähnliches „Deutungssymbol" auf dem Bildschirm entsprechend der Bewegungen der Maus auf seinem Schreibtisch steuern. Mit Hilfe dieses Pfeils kann er beispielsweise Funktionen aus einem Menü in einer sehr einfachen Weise auswählen: Er bewegt den Pfeil auf den Text (bzw. das Piktogramm) der Funktion und drückt auf den Knopf an der Maus. Da der Benutzer in diesem System nur *eine* Taste drücken muß, sind Fehleingaben weitgehend ausgeschlossen. Da er zusätzlich aber auch den Pfeil auf dem Bildschirm im Auge haben wird, kann er Reaktionen auf seine Funktionsauswahl auch sehr schnell erkennen. Dadurch ist die Maus das ideale Bedienelement für den ungeübten Computerbenutzer, da hier eine wesentliche bessere Übersichtlichkeit gegeben ist, als bei der manchmal recht verwirrenden Rechnertastatur.

Allerdings hat das Mauskonzept auch einen kleinen Nachteil: Das Positionieren der Maus braucht auch eine gewisse Zeit, tastaturgeübte Anwender könnten hier mittels Tasteneingaben schneller sein. Da die Maus jedoch hauptsächlich für ungeübte Computer-Anwender gedacht ist, fällt dieser Nachteil kaum ins Gewicht.

Im Folgenden wollen wir Ihnen das Wichtigste zum Thema Maus in einem Kurs aufzeigen. Da nicht jeder eine Maus besitzt, stellen wir zunächst ein Programm vor, das die programmtechnischen Abläufe einer Maus mit dem Joystick simuliert. Als Beispielprogramme sind eine Datenauswertung und ein Zeichenprogramm vorgesehen.

4/4.6.1

Maussimulation mit dem Joystick

Die bisherigen „Maus-Systeme“ sind alle für Personal-Computer der oberen Preisklasse konzipiert, was wohl darauf zurückzuführen ist, daß man hierfür einen grafikfähigen Computer braucht. Der Commodore 64 liegt zwar preislich weit unter den Personal-Computern, hat jedoch alle Voraussetzungen die zum Einsatz der Maus notwendig sind (Joystickanschluß, Grafik, Sprites). Aufgrund dieser idealen Hardwarevoraussetzungen ist die Implementierung einer Maus auf dem Commodore 64 nicht weiter schwierig. Die durch die ‚Sprites‘ gegebene Möglichkeit Objekte frei auf dem Bildschirm bewegen zu können, reduziert die Maus auf dem Commodore 64 auf folgendes System: Man nehme einen Sprite (kann wie ein Pfeil aussehen) und steuere diesen mit einer Maus bzw. mit einem Joystick auf dem Bildschirm. Damit ist bereits eine Minimal-Maus implementiert.

4/4.6.1.1

Grundlagen

Nachdem am Ende des vorigen Abschnittes schon kurz die Problematik der Maus-Software aufgezeigt wurde, wollen wir nun ein Maus-Steuerprogramm vorstellen, das auch mit einem Joystick verwendet werden kann.

Grundvoraussetzung ist natürlich, daß sich das Maus-Maschinenprogramm im Speicher befindet, entweder wird es als Maschinenprogramm geladen, oder aber von einem BASIC-Programm in den Speicher gePOKEd (siehe Beispiel-Listing).

Die Ansteuerung der Maus vom Anwenderprogramm aus geschieht über die Befehle PEEK, POKE und SYS. Dies hat den Vorteil, daß die Maus auch in Programmen verwendet werden kann, die später einmal compiliert werden (z.B. mit PETSPEED). Für die Anwendung ist allerdings die Kenntnis von einigen wichtigen Adressen notwendig, die hier angegeben werden.

Das Maus-Programm belegt 512 Bytes ab der Adresse 49152. Dieser Bereich darf deshalb vom Anwenderprogramm nicht überschrieben werden.

Um die Maus mit Parametern zu versorgen, wird ein Mouse-Control-Block (MCB) benutzt, der bei der Adresse 49643 beginnt. Folgende Werte werden im MCB übergeben:

```
MCB+00: Xmin (L,H)
MCB+02: Xmin
MCB+03: Xmax (L,H)
MCB+05: Ymax
```

In diese Register wird vom Anwenderprogramm aus geschrieben. Sie definieren ein Bildschirmfenster, das die Maus nicht verlassen kann. Die X-Koordinaten umfassen 2 Byte, da hier Werte von 0..319 möglich sind (Angabe in Hires-Koordinaten, hier ist die linke obere Ecke des Schirms gleich der Koordinate 0,0). Xmin, Ymin definieren die linke obere Ecke dieses Fensters, Xmax, Ymax die rechte untere Ecke.

```
MCB+06: OffsetX
MCB+07: OffsetY
```

Auch diese beiden Register müssen vom Anwenderprogramm aus versorgt werden. Sie enthalten den Offset in Rasterpunkten, um den die Pfeilspitze der Maus (oder generell der Punkt, mit dem gedeutet wird) von der linken oberen Ecke des entsprechenden Sprites entfernt sind. Dies hat die Ursache darin, daß der Benutzer die Form der Maus auf dem Bildschirm frei wählen kann. So kann er beispielsweise einen Pfeil oder aber ein Fadenkreuz als Deutungssymbol definieren. Es ist leicht einsehbar, daß die Pfeilspitze nicht an der selben Stelle innerhalb des Sprites liegen wird wie der Kreuzungspunkt beim Fadenkreuz. Mit Hilfe der Offset-Register wird diese Differenz jedoch wieder ausgeglichen.

```
MCB+08: Invert-Flag
```

Mit diesem Register kann man vom Anwenderprogramm aus die automatische Textinvertierung ein- bzw. ausschalten. Der Wert Null in diesem Register bedeutet, daß die Textinvertierung ausgeschaltet ist, jeder andere Wert schaltet sie ein. Achtung: Diese Funktion wird durch das Invertieren eines Textes unwirksam, d.h. dieses Flag wird vom Maus-Programm nach der Invertierung eines

Textes auf Null gesetzt. Die Invertierung kann übrigens mit SYS 49547 rückgängig gemacht werden.

MCB+09: Maus X (L,H)
MCB+11: Maus Y

Diese beiden Register sollen vom Anwenderprogramm nur gelesen werden, sie enthalten die momentane Position der Maus (in Hires-Koordinaten) auf dem Bildschirm.

MCB+12: Mausknopf

Auch dieses Register sollte nur gelesen werden. Es enthält den Status des Maus-Knopfes, der Wert Null bedeutet, daß der Knopf gerade nicht gedrückt ist. Will man vom Anwenderprogramm auf das Drücken des Knopfes warten, so genügt der Befehl WAIT MCB+12,1.

MCB+13: Textposition relativ zum Bildanfang (L,H)

Dieses Register hat nur einen Sinn, wenn vorher die automatische Textinvertierung eingeschaltet wurde. Sie enthalten die Startadresse des invertierten Textes relativ zum Bildanfang. Hierbei ist es völlig gleichgültig, auf welche Stelle innerhalb des Textes die Maus deutet, es wird der gesamte Text zwischen den zwei umschließenden Spaces invertiert und die Anfangsadresse des Textfeldes in diesem Register hinterlegt. Die Länge des invertierbaren Feldes ist jedoch auf 256 Zeichen begrenzt.

Nachdem nun der Mouse-Control-Block erklärt wurde, ist es natürlich wichtig zu wissen, wie die Maus aktiviert wird. Dazu muß der Anwender erst mal einen Sprite definieren, beispielsweise eine Pfeilspitze. Wir verwenden Sprite-0. Es müssen nun alle zur Sprite-Steuerung wichtigen Register mit Ausnahme des Sprite-Enable Registers und der Koordinatenregister im VIC versorgt werden. Danach sollte der MCB mit den Offset-Werten und dem Bildschirmfenster belegt werden.

Gibt man nun den Befehl SYS 49152 ein, so sollte sich die Maus mit einem am Control-Port-2 angeschlossenen Joystick steuern lassen. Hat man zudem die automatische Textinvertierung eingeschaltet (was auch nachträglich geschehen kann) und bewegt die Maus auf ein Textfeld, so sollte dieses beim Drücken auf den Joy-

stick-Knopf invertiert werden (d.h. in Negativdarstellung erscheinen). Ein weiteres Drücken des Knopfes hat jedoch keine Wirkung, da durch die Invertierung die Funktion wieder ausgeschaltet wird (was sehr sinnvoll ist, denn sonst würde das Textfeld bei längerem Drücken des Knopfes flackern, da es ständig invertiert wird).

Die Maus wird übrigens völlig unabhängig von den restlichen Computer-Operationen gesteuert, was durch eine Programmierung über Interrupt ermöglicht wurde. So kann man die Maus auf dem Bildschirm bewegen, während der Rechner beispielsweise den Bildschirminhalt verändert oder sogar während der Rechner Programme von Diskette liest. Die Geschwindigkeit der Maus liegt bei 60 Rasterpunkten/Sekunde, damit kann die Maus innerhalb von 6 Sekunden über den ganzen Schirm bewegt werden.

4/4.6.1.2

Listings

Zu dieser Beschreibung gehören zwei Listings, einmal der Quellcode für das Maus-Programm in einer für den in diesem Buch vorgestellten Assembler verständlichen Form, zum anderen ein kleines Demo-Beispiel, welches in BASIC geschrieben ist und das Maus-Programm in DATA-Zeilen enthält. Die Listings wurden jedoch nicht auf einem Commodore-Drucker erstellt, Cursor-Steuerzeichen tauchen daher nicht in der gewohnten Form auf, sondern als ASCII-Zeichen in Breitschrift. Wichtig beim Abtippen des Demo-Listings in BASIC: In Zeile 160 bis 180 sind die Spaces zwischen den Worten keine normalen Spaces, sondern geshiftete Spaces (also SHIFT und Space drücken). Das hat den Sinn, daß dann der ganze Satz vom Maus-Programm invertiert werden kann, da das geshiftete Space einen anderen Bildschirmcode hat als das normale Space.

Programm-Beschreibung des Assembler-Listings

Um eine möglichst hohe Geschwindigkeit und zugleich eine Unabhängigkeit von anderen Operationen zu erreichen, ist die Maus als Interrupt-Programm entworfen.

Der Commodore 64 erzeugt alle 1/60 Sekunden einen Interrupt, mit dem er die Tastatur abfragt und eine Software-Uhr weiterschaltet. Da diese Interrupt-Routine

über einen Vektor im RAM angesprungen wird ist es nicht weiter schwierig, eine eigene Interrupt-Routine einzuschleifen, die zusätzliche Aufgaben (hier die Maus-Steuerung) übernimmt.

Die Routine zum Ein- bzw. Ausschalten der Maus beginnt beim Label TOGGLE. Hier werden zuerst einmal ankommende Interrupts gesperrt, da ein Interrupt während der Manipulation am IRQ-Vektor den Rechner zum Absturz bringen könnte.

Nun werden die Koordinaten des Maus-Sprite aus den entsprechenden Registern des Video-Controllers gelesen und in Hires-Koordinaten umgewandelt. Dies geschieht durch die folgenden Rechenoperationen:

```
MausX: =SpriteX-24+OffsetX
MausY: =SpriteY-50+OffsetY
```

Die unterstrichenen Werte (24, 50) sind durch die Hardware bedingt (siehe C 64-Handbuch). Nun wird eine Routine namens HOLD aufgerufen, die die Funktion hat, die Maus innerhalb des definierten Fensters zu halten. Nachdem die Koordinaten-Transformation durchgeführt ist, wird das dem Sprite im Sprite-Enable-Register entsprechende Bit invertiert. Dadurch wird der Sprite beim ersten Aufruf der TOGGLE-Routine eingeschaltet, beim nächsten Aufruf wieder ausgeschaltet usw. Nun wird der Interrupt-Vektor auf die Maus-Routine „verbogen". Auch dies geschieht mittels Exclusive-OR-Verknüpfungen, dadurch wird der Vektor beim ersten Aufruf auf die Maus-Routine gebogen, beim nächsten Aufruf erhält er seinen ursprünglichen Wert usw. Nachdem nun der Interrupt-Vektor erfolgreich geändert wurde, werden mit einem CLI-Befehl die Interrupts wieder erlaubt. Nun kehrt das Programm in die Aufrufebene (meist BASIC) zurück, das Maus-Programm wird nun als Hintergrundprozeß alle 1/60 Sekunden abgearbeitet.

Nun folgt die bereits erwähnte HOLD-Routine. Hier werden die im MCB befindlichen Maus-Koordinaten mit den Koordinaten des definierten Fensters verglichen. Sollte sich die Maus außerhalb dieses Fensters befinden, so erhält die entsprechende Koordinate den Wert der nächstgelegenen Grenze. Allerdings bewirkt eine Untergrenze (des Fensters) von Null, daß die Maus beim Verlassen des Fensters auf der anderen Seite des Schirms wieder erscheint. Dies hat seine Ursache darin, daß für die Speicherung der Maus-Koordinaten nur absolute (positive) Werte verwendet werden. Nachdem die Überprüfung auf die Bereichs-

grenzen beendet ist, werden die Maus-Koordinaten (die in Hires-Form verliegen) wieder in Sprite-Koordinaten umgerechnet. Dies geschieht mit folgenden Rechenoperationen:

```
SpriteX: =MausX+24-OffsetX
SpriteY: =MausY+50-OffsetY
```

Bei der Umrechnung der X-Koordinate werden die CPU-Register X und Y als Zwischenspeicher verwendet. Im ersten Schritt wird die 24 addiert und das Ergebnis in X(L) und Y(H) gepuffert. Nun wird der Offset subtrahiert, das Low-Byte kann sofort in den Video-Controller geschrieben werden. Das High-Bit der Sprite-Koordinate wird mit Hilfe eines LSR-Befehls in das Carry-Flag übertragen. Nun wird das Video-Controller-Register, welches die High-Bits der Sprites enthält abhängig vom Carry-Flag verändert.

Dann folgt die eigentliche Maus-Steuerungs-Routine, also das Programmstück, welches alle 1/60-sec ausgeführt wird und die Maus steuert. Die erste Aktion dieser Routine ist das Rücksetzen des Maus-Knopf-Flags. Nun wird der Joystick-Port 2 abgefragt. Als erstes erfolgt die Abfrage, ob der Knopf gedrückt wird. Ist dies der Fall, dann wird das Feuerknopf-Flag auf den Wert 255 gesetzt, außerdem wird nun das Textinvertierungsflag überprüft. War es gesetzt, so wird das Unterprogramm CONVERT aufgerufen, welches für die Invertierung des Textes zuständig ist. Nun folgen die Abfragen auf die vier möglichen Richtungen des Joysticks. Sollte ein Bit für eine Richtung gesetzt sein, so wird die entsprechende Maus-Koordinate manipuliert. Nachdem die Koordinatenmanipulation abgeschlossen ist, wird wieder die bekannte Routine HOLD aufgerufen, die die Maus innerhalb des Bildschirmfensters fixiert. Schließlich wird die normale IRQ-Service-Routine des C 64 angesprungen.

Es folgt das bereits erwähnte Programmstück CONVERT. Die Aufgabe dieses Unterprogramms ist die Ermittlung der relativen Position der Maus zum Bildschirmanfang, also die Textposition, auf die die Maus gerade deutet. Hierzu wird folgende Formel angewendet:

```
TextPos: =INT(MausY/8)*40+INT(MausX/8)
```

Diese Formel hat ihren Ursprung in der 8*8 Rasterdarstellung der Zeichen. Um die Programmierung dieser Formel in Assembler möglichst einfach zu halten, wurde folgender Algorithmus verwendet:

```
MausY: =MausY AND 248
```

Diese boolsche Verknüpfung entspricht „INT(MausY/8)*8". Nun wird das Ergebnis zweimal um ein Bit nach links geschoben, also mit 4 multipliziert. Schließlich wird das ursprüngliche Ergebnis aufaddiert, somit ist der erste Teil der Formel berechnet. Nun wird das High-Bit der X-Koordinate mittels eines LSR-Befehls ins Carry transferiert. Im Low-Byte der Koordinate werden die untersten drei Bit mittels AND auf Null gesetzt. Dann folgt ein ROR, er schiebt alle Bits um eins nach rechts und berücksichtigt dabei auch das höchstwertige Bit im Carry. Nach 2 weiteren LSRs ist die X-Koordinate durch 8 dividiert und kann nun auf den bereits errechneten Teil der Textposition addiert werden.

Im nachfolgenden Programmteil wird zu dieser relativen Position die Startadresse des Video-RAMs addiert, man erhält die absolute Startadresse des Textes im Speicher. Von hier aus wird rückwärts nach dem nächsten Space (Code 32) gesucht. Enthält die Position selbst schon diesen Code, so wird die Invertierungsroutine umgehend verlassen. Die Suche endet sonst spätestens bei der Startadresse des Video-RAM's. Hat man durch das Auffinden des linken Begrenzungs-Space den Start des Textblocks gefunden, so werden anschließend alle Zeichen bis zum nächsten Space invertiert. Dies geschieht mit einer XOR-Funktion, die auf das Bit 7 im Video-RAM angewendet wird. Nach erfolgter Invertierung wird das Invertierungs-Flag gelöscht, um ein Flackern im 60 Hz-Rhythmus zu vermeiden. Die eigentliche Invertierungsroutine (ab Label REVERSE) kann auch vom Anwendungsprogramm direkt aufgerufen werden, z.B. um einen invertierten Text in sein ursprüngliches Aussehen zurückzuverwandeln.

Das Programmm befindet sich auf der Grundwerksdiskette

Programm-Beschreibung des BASIC-Listings

Dieses Listing ist für alle gedacht, die noch keinen Assembler haben. Zudem enthält es ein kleines Demo, aus dem die Programmierung der Maus recht gut ersichtlich ist.

Das Programmlisting gliedert sich in drei Blöcke, der Block bis Zeile 999 enthält das Hauptprogramm, der Block bis zur Zeile 10000 enthält diejenigen DATA-Zeilen, die sowohl die im Programm verwendeten Sprites definieren, als auch das Maschinenprogramm für die Maus-Steuerung enthalten. Der letzte Block enthält zwei Unterprogramme, die jeweils eine Informationsseite für eine Minute zur Anzeige bringen.

Interessant ist für uns jedoch nur der erste Block. In den Zeilen bis 100 werden die Sprite-Felder gefüllt. Außerdem wird das Maschinenprogramm aus den DATA-Zeilen gelesen und in den Speicher gePOKEd. Die Zeilen 100 bis 200 bauen den Menübildschirm auf, hier ist ein originalgetreues Abtippen die Grundvoraussetzung für das Funktionieren des Demoprogrammes. Die Leerzeichen zwischen den Wörtern in den Zeilen 160-180 müssen geshiftet sein (also SHIFT-Space eingeben). Die in Breitschrift gedruckten Zeichen sind, wie bereits erwähnt, Cursor-Steuerzeichen, ein breites „q“ entspricht z.B. Cursor nach unten.

Ab Zeile 300 wird es nun interessant. Hier wird nämlich das Fenster definiert, in dem sich die Maus später bewegen kann, außerdem der Offset auf die Mausspitze. Nun wird das Knopf-Flag sicherheitshalber auf Null gesetzt und die Maus mit einem SYS-Befehl aktiviert.

Anschließend wird die Textinvertierung erlaubt (POKE MC+8,1) und nun wartet das Programm auf das Drücken des Knopfes an der Maus (bzw. am Joystick).

In Zeile 345 wird die Bildschirmposition, auf die die Maus deutet, ermittelt und anschließend wird die Maus ausgeschaltet.

In den Zeilen 350 bis 360 wird abhängig von der ermittelten Position verzweigt, d.h. entweder das Programm beendet, oder es wird eines der Unterprogramme aufgerufen.

Zeigt die Maus auf keinen als Menüpunkt definierten Bildschirmbereich, so wird der Maus-Pfeil in Zeile 365 kurzzeitig auf ein „Fragezeichen“ umgeschaltet um dem Benutzer die Fehlbedienung zu signalisieren. Das Programm geht dann wieder in den Wartezustand.

Sollte das Programm bei Ihnen nicht richtig auf die Maus-Auswahl reagieren (Fragezeichen, obwohl ein Menüpunkt angewählt wurde), dann liegt das wahrscheinlich daran, daß Sie beim Abtippen des Menübildschirms einen Fehler gemacht haben. Am einfachsten ist es, wenn man zum Test nun eine Zeile 347 einfügt, in der die Variable „p“ ausgedruckt wird. Nachdem der Wert für „p“ bei jedem Menüpunkt ermittelt wurde, können die Zeilen 350 bis 360 entsprechend berichtigt werden. Danach kann man die Testzeile 347 natürlich wieder entfernen.

Somit wünschen wir Ihnen viel Spaß mit der Maus. Abschließend möchten wir noch auf die Ergänzungsausgaben zu diesem Buch verweisen, in denen weitere umfangreiche Anwendungsbeispiele für die Maus aufgeführt sind.

```
Data-Maus-Demo
==============

10 poke53281,1:poke53280,4:print"~"
20 printchr$(14);chr$(8);"S"
30 print" Schliessen Sie einen Joystick am"
40 print"q Port 2 des Rechners an und freuen"
50 print"q Sie sich auf......"
60 restore: rem sprites definieren
70 fori=0to62:reada:poke13*64+i,a:next
80 fori=0to62:reada:poke14*64+i,a:next
85 fori=49152to49664:reada:pokei,a:next
90 mo=49152:rv=49547:mc=49643:vc=53248
95 pokevc+39,2:poke2040,13:pokevc+23,0:pokevc+27,0:pokevc+29,0
99 fort=1to3000:next
100 print"S~"
110 print"            *** Die  SUPERMAUS  ***"
115 print"q            fuer Ihren Commodore-64"
120 print"qqV              Copyright (C) 1983 by"
140 print"q Z NATHANSOFT  Research Laboratories Europe"
150 print"\@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@"
160 print"~         Informationen ueber die Maus"
170 print"q          Wichtige Adressen der Maus"
180 print"q            ENDE DES DEMOPROGRAMMS"
190 print"\q@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@"
200 print"\ Waehlen Sie nun bitte eine Funktion an"
300 pokemc,40:pokemc+1,0:rem x-min
305 pokemc+2,88: rem y-min
310 pokemc+3,24:pokemc+4,1:rem x-max
315 pokemc+5,136:rem y-max
320 pokemc+6,1:rem offset-x
325 pokemc+7,1:rem offset-y
330 pokemc+12,0:rem fire-off
335 sysmo :rem maus-anschalten
340 pokemc+8,1:waitmc+12,1: rem warten auf knopf
345 p=peek(mc+13)+256*peek(mc+14):sysmo
350 ifp=649thenend
355 ifp=567thengosub20000:goto100
360 ifp=486thengosub10000:goto100
365 poke2040,14:sysmo:fort=1to250:next:poke2040,13:goto340
999 stop
1000 rem datazeilen fuer pfeil-sprite
1001 data 0,0,0
1002 data 64,0,0
1003 data 96,0,0
1004 data 112,0,0
```

4/4.6.1.3

Datenauswertung

Im Folgenden wollen wir Ihnen ein Datenauswertungsprogramm als Beispiel zur Verwendung unserer Joystick-Maus aufzeigen.

Zweck des Programmes

Um dieses Programm zu erklären, muß zuerst einmal eine Untersuchung des Hintergrundes abgegeben werden, der zur Erstellung dieses Programms geführt hat.

Der Autor ist begeisterter Sportschütze, und als solcher natürlich einem Verein angeschlossen. Dieser Verein hat folgende Regeln für seine internen Meisterschaften, die mit in dieses Programm eingebracht wurden:

- Jedes Vereinsmitglied hat eine Schützennummer, die zur Identifikation bei Namensgleichheit dient.
- Jeder Schütze kann über das ganze Jahr verteilt so viel Serien (40 Schuß im Normalfall) schießen, wie er will. Am Jahresende werden die 20 besten Ergebnisse für die Wertung der vereinsinternen Meisterschaft herangezogen.

Das Programm unterstützt nun die Berechnung der Meisterschaft, indem es alle im Jahresverlauf geschossenen Ergebnisse speichert. Selbstverständlich ist eine derartige Berechnung nicht erst am Jahresende möglich, sondern jederzeit, da man bereits während des laufenden Jahres gerne Vergleichsmöglichkeiten hat.

Weiterhin lassen sich natürlich alle Ergebnisse auflisten, und, was zur Leistungskontrolle besonders vorteilhaft ist: Alle Ergebnisse lassen sich auch grafisch auswerten. So sind Balken-Grafiken mit den Leistungswerten der einzelnen Monate genauso verfügbar wie eine Jahresübersicht, die die Mittelwerte der einzelnen Monate übersichtlich darstellt.

Die Dateneingabe erfolgt im Dialog, zudem ist das Programm durch einige Programmiertricks fast absturzsicher (Nach Murphy's Gesetzen gibt es keine vollkommen absturzsicheren Programme).

Damit das Programm auch von Computer-Laien bedient werden kann, wird die Steuerung voll über Menüs abgewickelt. Die Menüpunkte werden über die *Maus* angewählt, somit ist einfachste Bedienbarkeit gewährleistet.

Das Programm ist daher ein gutes Lehrbeispiel, bei dem vor allem der sinnvolle Einsatz der Maus aufgezeigt wird.

Programmtechnische Besonderheiten:

Dieses Programm besteht aus mehreren Teilprogrammen:
- dem Ladeprogramm
- dem Maus-Objektprogramm
- dem eigentlichen Auswertprogramm.

Das Ladeprogramm hat lediglich die Aufgabe, die einzelnen Programmteile (Maus-Objektcode und Auswertprogramm) einzuladen und zu starten. Wer bereits das Data-Maus-Demo aus diesem Buch abgetippt hat, kann es ganz einfach zu einem Ladeprogramm für eben diesen Zweck umfunktionieren, ansonsten sei auf die Beschreibung des Laders verwiesen.

Das Maus-Objektprogramm ist der Code, der vom Assembler erzeugt wird. Dieses Programm enthält also die 512 Bytes Maschinenroutinen, die für die Maus-Steuerung notwendig sind. Das Objektprogramm kann als PRG-File auf Disk stehen, oder aber als Data-Zeilen im Ladeprogramm.

Das Auswertprogramm ist schließlich die Anwendung, die vollständig in BASIC geschrieben ist. Allerdings ist das interpretierte BASIC nicht gerade sehr schnell. Das Programm wurde aber so ausgelegt, daß es von handelsüblichen Compilern (z.B. Petspeed 64) compiliert werden kann. Die compilierte Version ist natürlich um etliches schneller, beim Aufbauen von Balkengrafiken um den Faktor 20.

Das Ladeprogramm

Aus Gründen der Einfachheit beschreiben wir hier nur die Version des Ladeprogramms, welche auch das Maschinenprogramm für die Maussteuerung von Disk lädt.

Maschinenprogramme lassen sich relativ einfach von BASIC-Programmen aus einladen, es genügt ein LOAD-Befehl, bei dem als *Sekundäradresse* eine 1 angegeben wird. Diese Form des LOAD-Befehls bewirkt, daß das zu ladende Programm exakt an die Stelle geladen wird, die im Programmkopf der Diskette angegeben ist.

Ein LOAD-Befehl innerhalb eines BASIC-Programms bewirkt aber zugleich *immer* einen Sprung zur ersten Programmzeile. Das hat den Vorteil, daß man von einem BASIC-Programm aus ein anderes BASIC-Programm nachladen kann, dieses wird dann automatisch gestartet. Die Variablen des *aufrufenden* Programms werden dabei nicht zerstört, wenn folgende Bedingung eingehalten wird: Das nachgeladene Programm darf nicht länger sein als das Programm in dem der LOAD-Befehl steht.

Da beim Laden von Maschinenprogrammen der BASIC-Speicher nicht verändert wird, würde ein BASIC-Programm in einer Endlosschleife sein Dasein fristen, wenn man nicht programmtechnische Vorkehrungen trifft, die dies verhindern.

Aus diesem Grund ist die erste Zeile unseres Ladeprogramms bereits eine IF-Abfrage. Die Variable ML signalisiert, ob das Maschinenprogramm bereits geladen ist. Beim ersten Start des Programms, also unmittelbar nach dem RUN hat ML natürlich den Wert Null, das Programm arbeitet also die folgenden Zeilen ab. Hier folgt nun bald der Ladebefehl für das Maschinenprogramm, vorher bekommt ML aber noch den Wert 1 zugewiesen. Nachdem das Maschinenprogramm geladen wurde, startet der BASIC-Lader (der ja nicht verändert wurde) erneut. Nun erkennt er aber, daß das Maschinenprogramm bereits geladen ist und setzt seine Arbeit an der richtigen Stelle im Programm fort.

Als nächstes werden innerhalb des Ladeprogramms die Sprites für die Maus definiert, eine Aufgabe, wie wir sie bereits vom Maus-Demo her kennen.

Die Zeilen 90 und 99 bieten nun den eigentlichen Trick. In Zeile 99 wird das Auswertprogramm von Disk geladen, hier die compilierte Version (selbstverständlich kann man auch jeden anderen Namen einsetzen). Da dieses Programm den BASIC-Speicher überschreibt, startet es nach dem Laden automatisch (siehe Anmerkungen weiter oben). *Halt!* Jedem wird sicher aufgefallen sein, daß das Ladeprogramm um etliches kürzer ist als das Auswertprogramm, und das darf ja nicht vorkommen, da sonst keine Variablen übergeben werden. Variablen sollen auch gar nicht übergeben werden, aber selbst dann wäre eine solche Vorgehensweise nicht zulässig, da sich hier das geladene Programm ohne weiteres selbst zerstören kann. Um nun einen späteren Programmabsturz (durch Selbstzerstörung) zu verhindern, wurde die Zeile 90 eingebaut.

In dieser Zeile wird ein etwas hinterhältiger Trick angewendet, mit dem der BASIC-Interpreter überlistet wird. Der BASIC-Interpreter besitzt nämlich Zeiger auf den Start und das Ende eines Programms. Der Zeiger auf den Start steht in den Speicherstellen 43/44, der Zeiger auf das Ende in den Speicherstellen 45/46. Um nun dem BASIC-Interpreter vorzugaukeln, daß er ein sehr langes Programm im Speicher hat, muß man einfach den Zeiger auf das Ende etwas manipulieren. In unserem Beispiel geschah konkret folgendes:

Zuerst wurde die Länge des nachzuladenden Programmes ermittelt (Directory ansehen). In unserem Fall ist das nachzuladende Programm 72 Blöcke lang. Nun wird der Zeiger auf das BASIC-Ende einfach auf einen Wert gesetzt, der um 72 Pages (1 Page = 256 Byte) höher ist als der Zeiger auf den Anfang,

also mit *POKE 46, PEEK (44) + 72.* Die nachfolgenden CLR und RESTORE-Befehle sorgen dafür, daß alle anderen Zeiger zur Variablenverwaltung mit verändert werden, ohne daß man extra POKE-Befehle anwenden muß.

Mit dieser Methode läßt sich nun von dem relativ kurzen BASIC-Lader ein sehr viel längeres Hauptprogramm einladen und automatisch starten.

Somit können wir zur Beschreibung des Hauptprogrammes übergehen.

Die Startphase des Programms

In diesem Programmteil werden alle notwendigen Variablenfelder dimensioniert und, soweit erforderlich, mit Werten vorbelegt. So werden Felder für Monatsnamen und die zugehörige Anzahl der Tage aufgebaut; dies ist eine Hilfe, um später bei der Dateneingabe eventuelle Fehleingaben abzufangen (z.B. 45.03. als Datum). Das Feld BZ% enthält die Bildschirmcodes für die Balkengrafik. Wie wir später sehen werden, ist so eine Balkengrafik mit 200 vertikalen Abstufungen möglich, ohne die hochauflösende Grafik des VIC-Chips programmieren zu müssen.

Während des Programmstarts wird der Anwender nach dem aktuellen Datum gefragt, dieses taucht später in der Druckerauswertung wieder auf. Zudem wird die Datei mit den Ergebnissen eingelesen. Existiert noch keine Datei, dann wird eine neue Datei aufgebaut.

Hauptmenü Aufbau und Auswertung

In den Zeilen 410 bis 462 wird der Bildschirm mit dem Hauptmenü aufgebaut. Die variablen BF$ und B$ in den Textzeilen haben folgenden Zweck: Sie enthalten geshiftete Spaces, d.h. beim Anwählen mit der Maus werden sie bei eingeschalteter Textinvertierung invertiert. Mit dieser Methode kann man ganze Sätze, oder wie hier geschehen, ganze Blöcke mit 3 Bildschirmzeilen invertieren. Die Verwendung von Stringvariablen für die geshifteten Spaces hat ihren Grund in der Tatsache, daß der später verwendete BASIC-Compiler geshiftete Spaces in Texten falsch compiliert, er ersetzt sie durch „+“. Verwendet man hingegen Stringvariablen, dann läuft alles wunschgemäß.

In den Zeilen 470 und 475 wird nun die Maus aktiviert. Es werden die Fenster für die Maus definiert, und das Maus-Programm wird gestartet.

Zeile 476 wartet nun darauf, daß der Maus-Knopf gedrückt wird. Ist dies erfolgt, so schaltet Zeile 477 die Maus erst einmal ab.

In der Zeile 480 wird die Bildschirmposition des Textblockes berechnet, auf den die Maus beim Drücken des Knopfes zeigte. In den Zeilen 490 bis 510 wird nun entsprechend der angewählten Funktion verzweigt.

Kommt das Programm bis Zeile 511, so bedeutet dies, daß kein gültiger Menüpunkt angewählt wurde (das Programm würde sonst ja verzweigen). Somit wird die Maus kurz auf das Fragezeichensymbol umgeschaltet und das Programm geht nach Zurückschalten auf das Pfeilsymbol wieder in den Wartezustand über.

Programmendphase und Datenspeichern

Die Endsequenz des Programms beginnt bei Zeile 530. In Zeile 531 wird das Editierflag EF abgeprüft. Wurden nämlich während des Programmlaufs neue Serien eingegeben, dann muß die Datei neu gespeichert werden, wurde hingegen nur ausgewertet, so kann ein Speichern entfallen.

Ist ein Speichern der Daten notwendig, so geschieht dies in den Zeilen 535 bis 900.

An dieser Stelle sei auch kurz der Dateiaufbau beschrieben. Alle Ergebnislisten werden in SEQ-Dateien abgelegt. Die einzelnen Einträge sind untereinander mit Carriage-Return [CHR$(13)] getrennt. Die Einträge stehen in folgender Reihenfolge:

1. Die Anzahl der Serien, die in der Datei stehen.
2. Das Auswertungsjahr. Durch die Speicherung des Jahres kann man sich die Eingabe bei der Datenerfassung sparen.
3. Der Name des Schützen.

4. Die Schützenklasse (früher war hier einmal das Geburtsdatum angeführt, daher der etwas wenig assoziative Name GD$).
5. Die Schützennummer.

Ist die Anzahl der Serien größer als Null, so folgt nun eine Aufstellung der Serien und des dazugehörenden Datums. Das Datum wird *nicht* als String gespeichert, sondern ist in einer numerischen Variable folgendermaßen codiert:

Datumszahl = 32 x Monat + Tag

Der 25. 03. wäre also so verschlüsselt: 32x3+25=*121.* Diese Speichermethode spart Platz sowohl auf der Disk, als auch im Rechnerspeicher.

Editieren der Serientabelle

Menüaufbau und Auswertung

Hierzu ist eigentlich nichts Besonderes zu Sagen, auch hier läuft alles wieder nach dem bekannten Schema ab: Zuerst wird der Menübildschirm aufgebaut. Dann wird die Maus aktiviert, und nach dem Drücken des Knopfes wird abhängig von der errmittelten Bildschrimposition des Menüpunktes verzweigt.

Positionieren des Tabellenzeigers

Die Editierroutinen verwalten einen Tabellenzeiger ZE. Dieser wird beim Start der Editierfunktion auf die Tabellenmitte gesetzt, kann aber mittels Menüauswahl noch oben oder unten, oder sogar an den Anfang oder das Ende verschoben werden. Auch die Eingabe einer Position ist möglich. Zuständig hierfür sind die Zeilen 11000 bis 15050. Im Anschluß an eine Änderung des Zeigers wird der Tabellenausschnitt neu im Sichtfenster angezeigt.

Löschen eines Eintrags

Mit dieser Funktion kann der Tabelleneintrag gelöscht werden, auf den der Tabellenzeiger gerade zeigt. Hier wird noch eine Sicherheitsabfrage mit der Maus ausgeführt: Erst wenn der Benutzer hier „JA“ anwählt, wird die Löschung vorgenommen. Zudem wird hier das Editflag EF beeinflußt, es bewirkt dann ein Speichern der Datei beim Porgrammende.

Eingeben einer neuen Serie

Diese Funktion fordert zuerst die Ringzahl an, und gibt hier den Wert 0 vor. Drückt man nun RETURN, so wird die Funktion sofort verlassen, ein Eintrag in die Tabelle findet nicht statt. Wird hingegen ein Zahlenwert eingegeben, so wird das Datum erfragt. Beim ersten Mal erhält der Benutzer hier die Vorgabe „TT.MM.“, später den jeweils zuletzt eingegebenen Wert. Die eingegebene Serie wird im Folgenden in die Tabelle einsortiert, hierbei ist das Datum ausschlaggebend. Die Codierung des Datums als Zahl erleichtert hier wiederum die Sortierung. Bei Strings wäre die Bedingung, daß der 25.03. vor dem 01.04. kommt beispielsweise nicht gegeben, bei der Zahlencodierung hat der 01.01. den kleinsten Wert, und der 31.12. den größten. Das Programm sucht also die Stelle, an die die Serie eingefügt werden muß (Zeilen 16650 bis 16670) und verschiebt alle folgenden Eintragungen um eins nach hinten (Zeilen 16710 bis 16730). Der Tabellenzeiger zeigt dann auf die eingegebene Serie.

Bei der Eingabe ist noch folgende Besonderheit zu erwähnen: Das Programm benutzt nicht den normalen INPUT-Befehl sondern den Umweg über OPEN1,0: INPUT# 1,X$: CLOSE1. Der Umweg über eine logische Input-Datei hat den Vorteil, daß beim INPUT# kein Fragezeichen auf dem Schirm ausgegeben wird, wie man es vom normalen INPUT her kennt.

Tabellenausschnitt anzeigen

Das Programm zeigt einen fünfzeiligen Ausschnitt aus der Tabelle in einem Sichtfenster an. Hierfür ist die Routine ab Zeile 17000 zuständig. Die Serie, auf die der Tabellenzeiger deutet ist jeweils in der Mitte des Fensters, also die 3. Zeile im Fenster. Sollte der Zeiger auf den Anfang oder das Ende deuten, so werden die Zeilen darüber oder darunter selbstverständlich mit Leerzeichen gefüllt. Die Ausgabe der Daten erfolgt formatiert, die Formatierung wird über Stringfunktionen erreicht. Diese Funktionen kosten natürlich viel Zeit, und so kann es durchaus sein, daß das Programm hier etwas langsam wird.

Datenauswertung

Menüaufbau und Auswertung

Als Erstes wird hier geprüft, ob die Umsortierung der Serien (nach Ringzahl) notwendig ist. Das Programm verwaltet hierzu ein Flag namens SR, dieses wird von den Editierfunktionen „Eingeben“ und „Löschen“ gelöscht, und nach

erfolgter Sortierung von dieser Routine gesetzt. Immer wenn das Flag gelöscht ist, zeigt dies an, daß eine Neusortierung erforderlich ist. Grund für die Umsortierung ist die Tatsache, daß man zur Auswertung der Vereinsmeisterschaft die 20 besten Serien heranziehen muß.

Zugleich ist anzumerken, daß eigentlich nicht das Datenfeld sortiert, sondern vielmehr eine sortierte Liste mit Zeigern in das Datenfeld aufgebaut wird. Dies hat den Vorteil, daß das Feld nach Verlassen der Auswertfunktion nicht nochmals in das Speicherformat (nach Datum sortiert) umsortiert werden muß.

Nach eventuell erfolgter Sortierung geht es wie gewohnt weiter, ein Menübildschirm wird aufgebaut und mittels der Maus abgefragt.

Auswerten der Vereinsmeisterschaft

Hier wird eine Liste der 20 besten Serien auf dem Schirm angezeigt. Gleichzeitig werden von den angezeigten Serien Gesamtringzahl und Mittelwert berechnet. Schließlich wird noch ein kleines Menüfeld für die Maus definiert, einziger Menüpunkt ist hier das Verlassen der Funktion.

Anzeigen einer Gesamtliste

Diese Funktion ähnelt sehr stark der Auswertung der Vereinsmeisterschaft. Hier werden jeweils 30 Eintragungen auf dem Bildschrim angezeigt, zusätzlich wird ein Untermenü aufgebaut, in dem man vorwärts oder rückwärts blättern kann.

Balkengrafik Monate

Es wird ein Fenster aufgebaut, welches die Namen der Monate anzeigt. Mit der Maus wird der entsprechende Monat ausgewählt, die Programmierung erfolgt nach der bereits bewährten Methode.

Nun wird ein Koordinatensystem errichtet, welches in der vertikalen Achse beschriftet ist. Die Beschriftung richtet sich vornehmlich nach dem schlechtesten Ergebnis: Ist es beispielsweise 340, so ist die Beschriftung auf dem Niveau der horizontalen Achse „320“, also 20 weniger als das Minimum.

Nun werden die Balken aufgebaut. Dabei ist zu beachten, daß die Balken recht bunt sind, jeder Balken bekommt eine andere Farbe, die aus der Tabelle CO% geholt wird. Um einen Balken zu zeichnen, geht man in zwei Schritten vor:

Gegeben ist die Höhe des Balkens in Bildschirmpunkten, also z.B. 75.

Nun berechnet man, wieviel volle Bildschirmkästchen der Balken hoch ist, also INT(Höhe/8). Das Ergebnis wäre in unserem Beispiel der Wert 9. Das bedeutet, daß man die Basis des Balkens und die acht darüberliegenden Bildschirmkästchen mit dem Wert 160 vollPOKEd. 160 entspricht einem inversem Leerzeichen.

Anschließend wird berechnet, wieviel Bildschirmpunkte an Höhe noch fehlen, also der Rest der Division Höhe/8. In unserem Fall wäre dies 3. Nun wird das Zeichen BZ%(3) an die Stelle über dem letzten 160-er Kästchen gePOKEd. BZ%(3) enthält nämlich genau das Bildschirmzeichen, welches einem 3 Zeilen hohen Balken entspricht.

Mit dieser Methode ist es möglich, eine Balkengrafik mit Hires-Auflösung (200 Punkte vertikal) zu programmieren, ohne extra den VIC-Chip umständlich im Hires-Mode ansprechen zu müssen.

Die Routine erhält natürlich auch wieder einen mit der Maus anwählbaren Menüpunkt zum Verlassen der Funktion.

Balkengrafik Jahr

Hier wird ähnlich vorgegangen wie bei der Monatsgrafik. Es werden allerdings dickere Balken hergenommen, zudem wird eine Art 3-D-Darstellung gewählt. Dies sieht optisch sehr gut aus, hat aber den Nachteil, daß die Balken nur noch ganze Kästchen hoch sein können. Da die Balken jedoch nur Mittelwerte darstellen, fällt dieser Nachteil kaum ins Gewicht.

Auch hier wird die Maus wieder zum Verlassen der Funktion abgefragt.

Drucken einer Auswertungsliste

Diese Routine ist speziell auf das System des Autors zugeschnitten und muß daher im Bedarfsfall etwas abgeändert werden. Auf folgendem System läuft sie jedoch ohne Änderungen: Commodore 64 & Panasonic 1090 Drucker. Der Drucker ist über ein Kabel am User-Port angeschlossen, als Treibersoftware wird das Programm der Firma Brockhaus & Müller, Göttingen, verwendet. Ein kompatibles Schnittstellenprogramm ist in einer der folgenden Ergänzungen vorgesehen.

Im Anhang befindet sich ein Beispielausdruck, damit kann man sehr leicht feststellen, was alles gedruckt wird, und wie man das Programm eventuell an die eigenen Erfordernisse anpaßt.

Die Druckroutine kehrt selbständig wieder ins Auswertmenü zurück, eine Rückkehr durch Maus-Betätigung ist deshalb nicht notwendig.

Aufbauen einer neuen Datei

Diese Routine wird angesprochen, wenn man während der Startphase versucht hat, eine Datei zu laden, die noch nicht existiert.

Vom Benutzer werden nun die wichtigen Informationen für den Dateikopf erfragt. Im Anschluß daran wird die neue Datei auf Disk gespeichert. Das Programm setzt nun beim Hauptmenü seine Arbeit fort.

Diskettenfehlerbehandlung

Sollte es bei Diskettenoperationen einmal zu Fehlerzuständen kommen, so gibt diese Routine die Fehlermeldung der Disk-Station aus. Der Benutzer sollte dann den Fehler beheben (z.B. eine Diskette ins Laufwerk einlegen) und RETURN drücken.

Zeitverzögerung um 1/2 Sekunde

Diese Routine dient einzig dazu, bei Aufrufen von Menüpunkten mit der Maus diese eine kurze Zeit (1/2 sec) im invertierten Zustand zu belassen. Um hohe Zeitkonstanz zu gewährleisten, wird hier der 1/60-sec Timer TI abgefragt. Nachdem er auf 0 gesetzt ist, wird gewartet, bis er den Wert 30 (= 1/2 sec) erreicht hat, dann wird die Routine verlassen. Eine Zeitverzögerung hätte man zwar auch mit einer leeren FOR-NEXT-Schleife erzeugen können, aber spätestens nach dem das Programm Bekanntschaft mit einem Compiler gemacht hat, wäre jedem aufgefallen, daß dies nicht der optimale Weg ist (Compiler haben die nette Eigenart, FOR..NEXT sehr schnell zu machen, der Effekt der Zeitverzögerung ist dann nicht mehr gegeben).

4/4.7

Dateiverwaltung für den C 128

Nach dem Spieleinsatz und der Textverarbeitung werden die Home-Computer heute in den meisten Fällen zu Datenverarbeitungsproblemen herangezogen. Sogar Kleinbetriebe setzen Home-Computer zur Verarbeitung der betriebsinternen Daten ein. Durch die immer besser werdenden Hardwareeigenschaften und der allgemeinen Aufklärung über die Möglichkeiten von Computer, wächst auch der Bereich der Datenverwaltung in viele Einsatzgebiete hinein.

Während Kleinbetriebe ihre Lagerverwaltung bzw. die Pflege ihrer Kundenstammdaten als häufigstes Einsatzgebiet für einen Rechner betrachten, können im privaten Bereich mit einer Dateiverwaltung Dias, Schallplatten, Briefmarken und überhaupt alle Sammlungen verarbeitet werden. Aber auch die Adressen von Verwandten und Bekannten und deren Geburtstage lassen sich natürlich mit einem Computer speichern. Eine nach Geburtstagen sortierte Liste hilft sicherlich, manche peinliche Situation zu vermeiden.

Im Rahmen des Buches „Neue Möglichkeiten mit dem C 64/C 128" wollen wir selbstverständlich auch Dateiverwaltungsprobleme lösen. Wie wir eben erwähnt haben, sind diese unterschiedlichster Natur, und es wäre viel zu aufwendig, für jedes Problem eine eigene Dateiverwaltung zu schreiben. Auch die Standardsoftware ist den Weg weg von der speziellen Dateiverwaltung hin zu einer allgemeinen Dateiverwaltung gegangen. Diesen Weg wollen wir im Rahmen von Kapitel 4/4.7 nachvollziehen.

Aus verschiedenen Gründen bilden wir eine Dateiverwaltung nicht mehr auf dem C 64 ab, sondern bedienen uns dabei eines C 128. Einige der Kriterien sind die größere externe Speicherkapazität sowie die Möglichkeit der Verwendung eines 80-Zeichen-Bildschirms, der in der Regel bei Datenverwaltungen sinnvoller ist als ein 40-Zeichen-Bildschirm. Die umfangreichen Floppybefehle (siehe Kapitel 4/3.1.4) sind ein weiterer Grund für die Wahl des C 128. Dadurch brauchen wir uns nicht so sehr um die Diskettenorganiation unserer Daten zu kümmern. Ein C 128 läßt sich schon eher in einem Kleinbetrieb einsetzen als ein C 64.

Durch die größere Speicherkapazität ist es auch sinnvoller, kompliziertere Zugriffsverfahren als nur über die Nummer des Datensatzes zu verwenden. Dadurch können als Zugriffsschlüssel nicht nur relativ kleine Zahlen, sondern auch Namen oder Artikelnummern herangezogen werden.

Zunächst jedoch einiges zum Aufbau von Kapitel 4/4.7. Im ersten Unterkapitel werden wir auf theoretische Grundlagen eingehen, wie das Konzept der datenbeschreibenden Variablen, die Binär-Bäume sowie verkettete Listen, aber auch einige Hilfestellungen werden gegeben, damit das Programm auf den C 64 übertragen werden kann.

Einen großen Raum nehmen die benötigten Unterprogramme ein. Gegenüber den anderen in diesem Buch vorgestellten Listings wollen wir die verwendeten Unterprogramme getrennt vom Hauptprogramm besprechen. Unsere Programmierweise wird also hauptsächlich der Bottom-Up-Methode entsprechen, d.h. wir beginnen mit den wichtigsten Unterprogrammen und arbeiten uns langsam zu den Hauptprogrammen vor.

Ein weiterer Grund hat uns zu dieser Aufteilung veranlaßt: Die vorgestellten Unterprogramme können auch in eigenen Programmen verwendet werden, da wir zu jedem Unterprogramm die Ein- und Ausgabeparameter sowie verwendete „lokale" Variablen und einiges mehr dokumentieren werden. Betrachten Sie jedes Unterprogramm als kleines Utility. Gliedern werden wir die Unterprogramme in allgemeine Hilfsroutinen, Diskettenroutinen, Bildschirmroutinen, Druckroutinen, Binär-Baum-Routinen, Routinen für verkettete Listen und Funktionen.

Ein weiteres Unterkapitel ist einigen Übersichten zugedacht. Wegen der Komplexität der Gesamtdateiverwaltung dienen diese Übersichten dem allgemeinen Verständnis. Z.B. werden wir gleichbedeutende Variablen in verschiedenen Programmen einheitlich benennen. Auch die gesamten datenbeschreibenden, listenbeschreibenden und etikettenbeschreibenden Variablen — was wir später noch erläutern werden — sind in diesem Unterkapitel vorgestellt.

Kapitel 4/4.7.4 wendet sich dann den Dienstprogrammen zu, ohne die wir unsere allgemeine Dateiverwaltung nur schwer handhaben können.

Weitere Unterkapitel sind den Menüs, Hauptprogrammen, Selektier- und Sortierprogrammen sowie Druckprogrammen zugedacht.

In Kapitel 4/4.7.9 werden wir einige Datensatzbeispiele aufzeigen, um Ihnen die Einsatzmöglichkeiten der vorgestellten Dateiverwaltung zu demonstrieren.

Zum Schluß wollen wir Zusatzprogramme hauptsächlich für den kommerziellen Bereich vorstellen, wie z.B. eine Fakturierung, eine Offene-Posten Verwaltung, ein Mahnwesen, Möglichkeiten zur Lagerverwaltung sowie computerunterstütztes Bestellwesen.

Wenden wir uns jedoch den allgemeinen Eigenschaften unserer Dateiverwaltung zu. Wie bereits erwähnt, sollen viele Dateiverwaltungsprobleme damit gelöst werden, d.h. eine konkrete Beschreibung der verwendeten Datensätze ist nicht gegeben. Um einem Begriffswirrwarr vorzubeugen, hier zunächst einige kurze Erläuterungen zu den verwendeten Begriffen:

Dateiverwaltung	Gesamtkomplex eines Datenverwaltungsprogrammes.
Datensatz Record	Struktur zur Speicherung gleichartiger Daten. Ein Datensatz kann einem Kunden, einem Artikel oder einer Schallplatte zugeordnet sein. Der Datensatz sollte im Rahmen der Soft- und Hardwaremöglichkeiten das durch ihn beschriebene Objekt ausreichend charakterisieren.
Datei	Unter dem Begriff Datei wollen wir im Folgenden die Zusammenfassung aller gleichartigen Datensätze verstehen.
Datzensatz-element, Item	Die einzelnen Teile eines Datensatzes werden Element oder Item genannt. Bei einem Datensatz, der eine Adresse abbildet, wären z.B. Datensatzelemente durch die Begriffe Name, Vorname, Straße, Postleitzahl, Ort, Telefon bezeichnet. Generell ist zwischen dem Namen der Datensatzelemente (wie vorher aufgezählt) und dem Inhalt zu unterscheiden. Auf diese Problematik werden wir in Kapitel 4/4.7.1.1 noch näher eingehen.
Relative Datei, Direktzugriffs-datei	Das Betriebssystem des C 128 — und auch das des C 64 mit einigen Tricks — läßt sogenannte relative Dateien oder Direktzugriffsdateien zu. Im Gegensatz zu sequentiellen Dateien kann man gezielt auf einen bestimmten Datensatz zugreifen. Unsere gesamte Dateiverwaltung basiert auf relativen Dateien. Diese relativen Dateien benötigen als Zugriffsschlüssel eine Datensatznummer, wie wir später noch sehen werden. Um unsere Dateiverwaltung jedoch auch bei dem Zugriffsschlüssel allgemein zu halten, werden wir aus dem ersten Element des Datensatzes einen Zugriffsschlüssel errechnen. Dies geschieht mit Hilfe eines bestimmten Suchalgorithmuses.
Binärbäume	Ein Verfahren zum Suchen von Daten, das wir in unserer allgemeinen Dateiverwaltung verwenden werden. Auf die theoretischen Grundlagen werden wir in Kapitel 4/4.7.1.2 näher eingehen und die praktische Durchführung im Rahmen der Unterprogramme behandeln.
Verkettete Listen	Ein weiteres Verfahren zum schnellen Auffinden von Daten. Auch die Theorie der verketteten Listen werden wir in einem eigenen Kapitel (4/4.7.1.3) behandeln.

Wie Sie sehen, haben wir einiges vor. Natürlich werden wir deshalb das Thema nicht in ein oder zwei Ergänzungsausgaben vollständig abhandeln können. Wir werden zuerst mit einigen Dienstprogrammen beginnen und das Projekt weiterhin so aufbauen, daß Sie möglichst schnell mit Ihrer Dateiverwaltung arbeiten können. Nach den Grundlagen im Hauptprogramm wird das System ständig erweitert.

Wir beginnen mit dem Konzept der datenbeschreibenden Variablen und einem Dienstprogramm zum Anlegen dieser Daten.

4/4.7.1

Theoretische Grundlagen

Da man zum Verständnis des Gesamtkomplexes auch ein wenig Theorie braucht, haben wir diese der handwerklichen Tätigkeit, sprich Programmierung, vorangestellt. Zunächst wollen wir dabei das Konzept der datenbeschreibenden Variablen vorstellen, dann auf die theoretischen Grundlagen der Binärbäume und verketteten Listen eingehen. Weiterhin wollen wir Ihnen eine Hilfestellung geben, wie Sie das vorgestellte Programm auf Ihren C 64 umschreiben können.

4/4.7.1.1

Konzept der datenbeschreibenden Variablen

Wie wir bereits im Vorspann zu Kapitel 4/4.7 erwähnt haben, wollen wir unsere Datensatzprobleme unabhängig von der Datenstruktur bewältigen. Wir müssen also bei der Programmierung davon ausgehen, daß wir nicht wissen, wie viele Datenelemente ein Datensatz enthält und wie diese selbst strukturiert sind. Sind es Zeichenreihen, Zahlen oder Kalenderdaten? Wie lang dürfen die Zeichenreihen oder wie groß die Zahlen sein?

Unsere Dateiverwaltung muß also für den jeweiligen konkreten Anwendungsfall in der Lage sein, die vom Benutzer gewünschten Daten in der gewünschten Art und Weise zu speichern.

Normalerweise würde man z.B. bei einer Adreßverwaltung sechs Datenelemente vorsehen, die die Bezeichnung Name, Vorname, Straße, Postleitzahl, Ort und Telefonnummer haben. Den Datenelementen eines Datensatzes in dieser Dateiverwaltung würde man bei Programmierung für dieses konkrete Problem vielleicht die Variablennamen NA$, VN$, S$, PL$, O$ und TE$ geben. Die Bezeichnung der einzelnen Datenelemente am Bildschirm beim Erfassen oder Anzeigen würde dann per Programm mittels eines PRINT-Befehls vorgenommen werden müssen.

Will der gleiche Anwender aber auch seine Schallplatten mit einer Dateiverwaltung verarbeiten, so müßte er ein weiteres — neues — Programm schreiben, das Datensätze mit den Elementen Titel, Interpret, Spieldauer und Bemerkungen enthält. Aus mnemotechnischen Gründen würden hier vielleicht die Bezeichnung TI$, IN$, SD$ und BE$ verwendet. Auch alle PRINT-Ausgaben müßten neu formuliert werden.

In den oben genannten Fällen beschreiben also Variablen wie NA$, VN$ oder TI$ und IN$ die Inhalte eines einzelnen Datensatzes. Diese Stufe gilt es für eine **allgemeine** Dateiverwaltung zu abstrahieren. Die erste Abstraktionsstufe wäre es, die Anzahl der Elemente je Datensatz zu verallgemeinern. Was nimmt man her, wenn man Daten verarbeiten will, deren Werte bei der Programmierung noch nicht bekannt sind: Variablen.

Bei einer allgemeinen Dateiverwaltung benötigen wir also Variablen, die den Inhalt anderer Variablen beschreiben. Neben den eigentlichen zu speichernden Daten der Dateiverwaltung muß also vorher ein Schritt erfolgen, in der die Struktur des Anwendungsfalles ebenfalls in Variablen festgehalten wird. Dies ist ein kleiner Nachteil gegenüber Dateiverwaltungen, die sich mit einem konkreten Problem beschäftigen, jedoch wertmäßig nicht mit den Vorteilen zu vergleichen. Statt der Neuprogrammierung einer Dateiverwaltung brauchen Sie lediglich die allgemeine Dateiverwaltung heranzuziehen und eine Datei mit einer neuen Struktur aufzubauen. Dazu schreibt man sich sinnvollerweise ein kleines Dienstprogramm, wie wir es in Kapitel 4/4.7.4.1 vorstellen.

Im folgenden sollten sie also immer im Hinterkopf behalten, daß wir bei unserem Projekt mit zwei verschiedenen — logischen — Variablenebenen arbeiten. Wir unterscheiden logisch zwischen datenbeschreibenden Variablen, und Variablen, die die zu verwaltenden Daten aufnehmen.

Anhand der wichtigsten Variablen für die Datenstruktur in unserer Dateiverwaltung wollen wir Ihnen im folgenden kurz die Vorgehensweise aufzeigen. Außer der Datenstruktur werden wir in unserem Projekt auch das Aussehen von Listen- und Etikettenvariablen flexibel halten, wobei wir uns des gleichen Schemas bedienen.

Wenn ein Anwender die Datenstruktur für eines seiner Probleme eingeben möchte, was müssen wir dann von ihm wissen? Zunächst brauchen wir die Anzahl der Elemente pro Datensatz. In unserem Fall sehen wir hierfür die Variable AW (Anzahl der Werte) vor. Weiterhin brauchen wir für jedes einzelne Datensatzelement eine Beschreibung, um die Tätigkeiten am Bildschirm bedienungsfreundlich zu gestalten, und um den Listen in den betreffenden Spalten eine Überschrift zu geben. Wir wollen dies als Namen eines Elementes bezeichnen und benutzen die Variable NA$.

Da jedes Datensatzelement einen eigenen Namen erhält, müssen wir also ein Feld NA$(AW) dimensionieren. Außerdem müssen wir von jedem Datensatzelement noch wissen, ob es sich um Zahlen, Zeichenreihen oder Kalenderdaten handelt. Kalenderdaten müssen deshalb getrennt aufgeführt werden, da sie weder als Zeichenreihe noch als Zahl in der zu erfassenden Form zu sortieren sind. Außerdem

sind hier andere Plausibilitätsprüfungen erforderlich. Für die Art der Datenelemente sehen wir ein Feld AR$(AW) vor, das folgende Einträge enthalten kann:

a :	Alphanumerisch (Zeichenreihe)
i :	Ganze Zahlen (Integer)
f :	Fließkommazahlen
d :	Kalenderdaten

Wegen der Besonderheit der Fließkommazahlen geben wir bei der Art auch an, um wie viele Vor- und Nachkommastellen es sich maximal bei dem gewünschten Eintrag handelt. ‘f6.1’ bedeutet dabei, daß die Zahl Werte mit maximal sechs Vorkomma- und einer Nachkommastelle hat.

Für eine korrekte Speicherung wird als letztes noch die Länge jedes einzelnen Datensatzelementes benötigt, was dann — inklusive der Trennzeichen innerhalb des Datensatzes auf der Diskettenstation — die Gesamtlänge des Datensatzes ergibt. Beachten sollte man bei der Festlegung der Datenstruktur, daß die Felder für das größtmögliche Vorkommen eines Dateninhaltes dimensioniert sein müssen. Dies bedeutet nicht, daß man ein Datensatzelement zur Aufnahme eines Straßennamens mit 40 Zeichen festlegen muß, weil es solch ausgefallene Straßennamen gibt. Solche Ausnahmen lassen sich nicht nur bei Straßennamen sinnvoll abkürzen. Ein vernünftiger Mittelweg läßt sich also in den meisten Fällen — besonders bei Zeichenreihen — finden. Bei Datensatzelementen, die Zahlen beinhalten, sieht die Sache etwas anders aus: Wer verzichtet schon gerne auf Umsatz, nur weil der Computer an der entsprechenden Stelle keine größeren Zahlen aufnehmen kann? Aber auch für spätere Änderungen der Datenstruktur wollen wir uns ein Dienstprogramm basteln.

Postleitzahlen sollten als Zeichenreihen vereinbart werden, da hier kein Vorzeichen anfällt und auch kein Rechnen mit diesen Zahlen erforderlich ist. Auch Artikelnummern, die nur aus Ziffern bestehen, sollten als Zeichenreihe definiert werden. Insbesondere, wenn über die Artikelnummer zugegriffen wird (Zugriffsschlüssel in unserer allgemeinen Dateiverwaltung ist das erste Datensatzelement), muß diese als Zeichenreihe deklariert sein.

Bei ganzen Zahlen ist zur gewünschten Größe des absolut möglichen Höchstbetrages (9,99,999, . . .) eine weitere Stelle für das Vorzeichen hinzuzufügen. Für Eintragungen bis zur Höhe von 999 ist eine Länge von vier Einheiten vorzusehen. Die Länge von Kalenderdaten ist generell mit 8 Zeichen (TT.MM.JJ) zu veranschlagen und die Länge von Fließkommazahlen ergeben sich aus der Eingabe des entsprechenden Elementes AR$(). Zu den Vor- und Nachkommastellen werden hier jeweils noch zwei Einheiten für den Dezimalpunkt und das Vorzeichen hinzugezählt. Ein Datensatzelement, das maximal 999999,9 enthalten kann, belegt somit eine Länge von neun Einheiten.

Generell werden wir in unserer Dateiverwaltung nur mit Zeichenreihen arbeiten, d.h. daß insbesondere auch Zahlen als Zeichenreihe dargestellt werden. Dies vereinfacht

das Speichern auf und Lesen von Diskette erheblich, wie wir später noch sehen werden.

Generell ist es in der Datenverwaltung so, daß höchst selten mit Zahlen gearbeitet werden muß. Sofern mit den erfaßten Zahlen gerechnet werden soll, werden diese über die VAL-Funktion aus der jeweiligen Zeichenreihe ermittelt, und Ergebnisse können mittels der Funktion STR$ wieder in eine Zeichenreihe umgewandelt werden.

Die Bedeutung weiterer beschreibender Variablen sind aus den Übersichten in Kapitel 4/4.7.3 ersichtlich.

4/4.7.1.2

Binär-Bäume

Relative Dateien bieten zwar den Vorteil des Direktzugriffes auf einen speziellen Datensatz, um diesen Datensatz nach bestimmten Kriterien aber zu finden, bleibt einem normalerweise nichts anderes übrig als eine sequentielle Suche, es sei denn, man verwendet spezielle Suchverfahren, die zwar ein Mehr an Programmierarbeit benötigen, das spätere Suchen jedoch effizienter gestalten.

Die bekanntesten Suchverfahren, die in Datenbanken jeglicher Größe verwendet werden, sind die Suchbäume und in bestimmten Fällen das Hashing. Für die Datenmengen, die sich mit einem C 128 samt Floppy-Laufwerken speichern lassen, reichen Binär-Bäume vollkommen aus. AVL-Bäume sind eine spezielle Form dieser Binär-Bäume, die einen gewichteten Binär-Baum (darauf werden wir später noch eingehen) zur Grundlage haben. Eine Weiterentwicklung sind die B*-Bäume, die in ihren Knoten mehrere Bewertungen aufnehmen. Letztere werden insbesondere bei den größten Datenbanken eingesetzt.

Nachdem Sie jetzt so viele neue Begriffe kennengelernt haben, wollen wir etwas Ordnung in dieses Wirrwarr bringen und das Wesen der Suchbäume erläutern.

Genau wie in der Natur sind auch Suchbäume vielfach verästelt. Binär-Bäume sind eine Sonderform, wo bei jeder Teilung nur zwei Zweige entstehen, daher der Name Binär-Bäume. Der Stamm teilt sich also nur in zwei Äste, ein Ast nur in zwei Ästchen, ein Ästchen nur in zwei Zweige usw. Anders als in der Natur werden allerdings Suchbäume mit der engsten Stelle (Spitze) oben dargestellt und verästeln sich nach unten. Auch heißt der grundlegende Eintrag in einem Suchbaum nicht Stamm sondern Wurzel.

Die wichtige Information befindet sich auch nicht auf den Ästen und Zweigen, sondern an den Verteilerstellen, Knoten genannt. Die Verbindungen zwischen den Knoten werden durch sogenannte Zeiger dargestellt. Da die Sprache nicht allzuviele Möglichkeiten für die Wertigkeit eines Knotens im Baum zur Verfügung stellt (Stamm, Ast, Zweig, Blatt) betrachtet man in der Regel immer einen Knoten im Binär-Baum und spricht von dessen Vater, der näher an der Wurzel liegt, und von Söhnen, die weiter von der Wurzel entfernt liegen. Da es bei einem Binär-Baum nur zwei Söhne geben darf, spricht man auch von einem linken und rechten Sohn. Welche Bewandtnis es damit auf sich hat, werden wir später erfahren. Hier zunächst eine Zusammenfassung der wichtigsten Begriffe:

Binär-Baum:	Art eines Suchbaumes zum schnellen Auffinden von Daten.
Wurzel:	Erster Eintrag in einem Binär-Baum. Jede Suche beginnt bei der Wurzel. Die Wurzel hat keinen Vater.
Vater:	Höherwertiger Knoten im Binär-Baum (liegt eine Stufe näher an der Wurzel). Jeder Knoten — außer der Wurzel — hat **genau einen** Vater.
Sohn:	Nachfolger eines aktuellen Knotens in Richtung entgegengesetzt der Wurzel. Knoten im Binär-Baum können keinen, einen oder zwei Söhne haben.
Knoten:	Stelle im Binär-Baum die eine Bewertung (Zugriffsschlüssel) enthält. Die Zusammenfassung aller Knoten ergibt den Binär-Baum.
Blatt:	Knoten im Binär-Baum ohne Söhne.
Teilbaum:	Teil eines Binär-Baumes, der ab einem bestimmten Knoten alle dessen Söhne, Enkel usw. enthält.
Enkel:	Sohn des Sohnes eines Knotens (zwei Ebenen entgegengesetzt der Wurzel).
Großvater:	Vater des Vaters (zwei Ebenen näher zur Wurzel).
Ebene:	Alle Knoten, die von der Wurzel die gleiche Entfernung haben, bilden eine Ebene.
Bruder:	Zwei Knoten auf der gleichen Ebene, die den **gleichen** Vater haben, sind Brüder.
Linker Sohn:	Sohn, dessen Bewertung kleiner ist als die des Vaters.
Rechter Sohn:	Sohn, dessen Bewertung größer ist als die des Vaters.
Knotennummer:	Nummer eines Knotens, die mit der Nummer des Datensatzes identisch ist, unter der einerseits der Knoten in der Datei gespeichert ist, andererseits die Stammdaten zu finden sind.
Zeiger:	Verbindungen zwischen den Knoten eines Suchbaumes, die wir durch Zahlen (die Knotennummern) darstellen wollen. Diese Zahlen lassen sich auch im Rechner speichern und gestatten uns somit das Abbilden eines Binär-Baumes im Rechner.
Bewertung:	Wert des Knotens im Suchbaum.
Zugriffsschlüssel:	= Knotenbewertung. Über diesen Zugriffsschlüssel sucht der Anwender den gewünschten Datensatz.
Key:	Kurzbezeichnung für den Zugriffsschlüssel.

In Bild 4/4.7.1.2-1 haben wir das Schema eines Knotens aufgezeigt, wie wir es im folgenden verwenden wollen. Als wichtigstes haben wir in der Mitte die Knotenbewertung eingetragen. In den vier Eckfeldern bringen wir die Knotennummer (= Datensatznummer) unter und die Zeiger auf den Vater sowie die Söhne. Der Zeiger auf den Vater wird oben untergebracht, um die bildliche Darstellung zu vereinfachen und die beiden Söhne sind entsprechend ihrer Bezeichnung angeordnet.

<table>
<tr><td>Nummer</td><td>Vater</td></tr>
<tr><td colspan="2">Bewertung</td></tr>
<tr><td>Linker Sohn</td><td>Rechter Sohn</td></tr>
</table>

Bild 4/4.7.1.2-1 Schema eines Knotens im Binär-Baum.

In den folgenden Kapiteln werden wir die einzelnen Bearbeitungsschritte im Zusammenhang mit dem Binär-Baum aufzeigen.

4/4.7.1.2.1

Suchen und Einfügen

Um einen Binär-Baum zu erhalten, müssen wir zwei grundsätzliche Regeln **immer** beachten:

1) Die Bewertung eines linken Sohnes muß immer kleiner als die Bewertung des Vaters sein und die Bewertung eines rechten Sohnes muß immer größer sein als die Bewertung des Vaters.

2) Jede Bewertung darf nur einmal auftreten (Eindeutigkeit).

Als Bewertung in unserer allgemeinen Dateiverwaltung verwenden wir **immer** den Eintrag im ersten Datensatzelement. Das erste Datensatzelement ist also immer unser Zugriffsschlüssel, was bei dem Festlegen einer Datenstruktur zu berücksichtigen ist.

Mit diesem Wissen können wir uns nun an das Suchen und Einfügen einer Bewertung in dem Binär-Baum machen. Grundsätzlich ist jedem Einfügen ein Suchen vorauszuschicken, was gleich zwei Zwecke erfüllt: Einerseits wird so sichergestellt, daß kein Key doppelt auftritt, andererseits stellen wir durch das Suchen fest, wo der neue Eintrag im Baum anzuhängen ist. Die entsprechenden Unterprogramme in unserer Dateiverwaltung finden Sie ab Zeile 20000 und 20500 sowie ab 21000.

Für die folgenden Beispiele wollen wir annehmen, daß als Zugriffsschlüssel/Bewertung Namen vorgesehen sind. Mit einem kleinen Hilfsprogramm, was wir später erläutern werden, werden die Daten des jeweils gespeicherten Binär-Baumes im folgenden ausgegeben.

Zunächst brauchen wir die Wurzel. Diese wollen wir mit „Schneider Hans Lorenz" benennen. Nach der Erfassung ist in der Datei „Kundenkey" der in Bild 4/4.7.1.2.1-1 dargestellte Eintrag vorhanden.

Datei: Kunden	Anzahl der Datensätze: 1			
Bewertung	Nummer	Links	Rechts	Vater
Schneider Hans Lorenz	2	0	0	0

Bild 4/4.7.1.2.1-1 Datei Kundenkey nach Aufnahme der Wurzel

An dieser Stelle noch gleich eine Anmerkung: Wie Sie vielleicht bei den Programmteilen schon festgestellt haben, müssen wir die aktuelle Anzahl der Datensätze auch irgendwo zwischenspeichern. Da es eine Datensatznummer 0 bei den relativen Dateien nicht gibt, wählen wir uns dazu den ersten Datensatz aus, d.h. die eigentlichen Datensätze beginnen erst bei der Datensatznummer 2. Dies trifft sowohl auf die Dateien <Name>da1 und <Name>da2 zu als auch auf die Binär-Baum-Datei <Name>key. Unter „Nummer“ finden Sie also sowohl die Nummer des Datensatzes als auch die Knotennummer, die identisch sind. Unter „Links“ und „Rechts“ finden Sie die Zeiger auf die entsprechenden Söhne und unter „Vater“ den Zeiger auf den Vater.

Wie gesagt, wollen wir die Zeiger durch Zahlen darstellen, die die jeweilige Knotennummer des Sohnes oder des Vaters angeben.

Wenn wir nun unser Schema aus Bild 4/4.7.1.2-1 heranziehen, erhalten wir den in Bild 4/4.1.2.1-2 dargestellten minimalen Binär-Baum, der nur aus der Wurzel besteht. Nicht vorhandene Zeiger wollen wir durch die Zahl Null darstellen. Die Wurzel hat naturgemäß keinen Vater, woraus die Null im entsprechenden Feld resultiert, und Söhne sind zur Zeit auch noch nicht vorhanden.

2	0
Schneider Hans Lorenz	
0	0

Bild 4/4.7.1.2.1-2 Binär-Baum nach Festlegen der Wurzel.

Diesem Umstand wollen wir nun abhelfen: Der Eintrag „Eberl Werner“ ist in dem Baum neu einzufügen. Vorgehensweise: Zuerst suchen, ob dieser Eintrag noch nicht vorhanden ist. Im Moment mag Ihnen dies etwas unsinnig scheinen, aber einerseits müssen wir wissen, wo der neue Eintrag anzuhängen ist, andererseits ist es eine Übung für später.

Wir beginnen unsere Suche bei der Wurzel. Wir stellen fest, daß die neue Bewertung kleiner ist als die der Wurzel. Nach unserer Regel (1) ist also der linke Sohn der Wurzel für den weiteren Verlauf zuständig. Da hier eine Null eingetragen ist, stellen wir fest, daß eine Bewertung, die gleich der einzufügenden Bewertung ist, noch nicht vorhanden ist. Wir merken uns aber den zuletzt bearbeiteten Knoten (in diesem Fall die Wurzel (KI=2)).

Außerdem müssen wir die Anzahl der Datensätze (Records) jetzt erhöhen. Bisher hatten wir einen Datensatz, nun müssen wir uns zwei Datensätze merken und verwenden die Variable AR, die wir auf zwei setzen. Da wir den ersten Datensatz nicht beschreiben, ist die neue Knotennummer AR+1=3. Wir kennen also jetzt schon den Zeiger auf den Vater (2) und die Knotennummer (3). Söhne hat ein neu angefügter Knoten natürlich nie, da wir nach obigem Schema nur unten am Baum anfügen und nicht zwischendrin. Ein Anfügen mitten im Baum wäre auch programmtechnisch viel aufwendiger und würde auch mehr Rechenzeit kosten.

In der Datei <Name>key müssen wir natürlich nicht nur die Daten des neuen Knotens eintragen, sondern auch den neuen Knoten als Sohn bei dessen Vater aufhängen. Das schematische Ergebnis sehen Sie in Bild 4/4.7.1.2.1-3 und den Dateiauszug in Bild 4/4.7.1.2.1-4.

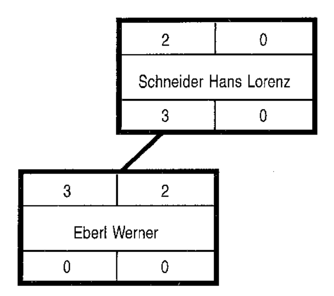


Bild 4/4.7.1.2.1-3 Binär-Baum nach Einfügen von „Eberl Werner".

Datei: Kunden	Anzahl der Datensätze: 2			
Bewertung	Nummer	Links	Rechts	Vater
Schneider Hans Lorenz	2	3	0	0
Eberl Werner	3	0	0	2

Bild 4/4.7.1.2.1-4 Datei Kundenkey nach Aufnahme von „Eberl Werner".

Als nächstes wollen wir die Bewertung „Friese Manfred" in den Binär-Baum aufnehmen. Dazu durchsuchen wir unseren bisherigen Binär-Baum (Bild 4/4.7.1.2.1-3) und stellen fest, daß die neue Bewertung wiederum kleiner ist als die Bewertung der Wurzel. Jetzt haben wir bereits aber einen linken Sohn (der für die kleineren Bewertungen zuständig ist) und prüfen als nächstes diesen linken Sohn. Dort stellen wir fest, daß die einzufügende Bewertung größer ist als „Eberl Werner" und prüfen den rechten Sohn (der für die größeren Bewertungen zuständig ist). Hier stellen wir eine Null fest und wissen dadurch, daß eine Bewertung „Friese Manfred" noch nicht im Binär-Baum enthalten ist.

Außerdem merken wir uns die Nummer des zuletzt geprüften Knotens (KI=3), erhöhen die Anzahl der Datensätze (AR=3) und fügen den neuen Knoten in den

Binär-Baum ein, wie es Bild 4/4.7.1.2.1-5 veranschaulicht. Bei diesem Bild haben wir auch die verschiedenen Ebenen angedeutet. In Bild 4/5.7.1.2.1-6 finden Sie den zugehörigen Auszug von der Diskette.

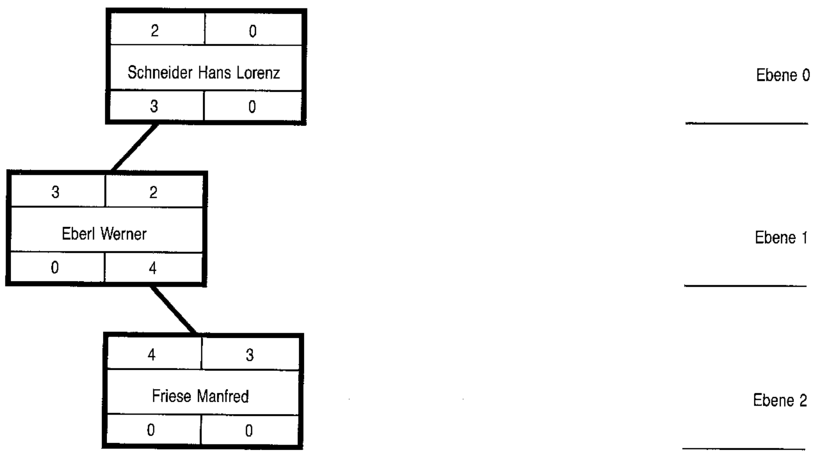


Bild 4/4.7.1.2.1-5 Binär-Baum nach Einfügen von „Friese Manfred".

Datei: Kunden	Anzahl der Datensätze: 3			
Bewertung	Nummer	Links	Rechts	Vater
Schneider Hans Lorenz	2	3	0	0
Eberl Werner	3	0	4	2
Friese Manfred	4	0	0	3

Bild 4/4.7.1.2.1-6 Datei Kundenkey nach Aufnahme von „Friese Manfred"

Die nächste einzufügende Bewertung soll „Kohl Klaus" sein. Wir beginnen unsere Suche wieder bei der Wurzel und stellen fest, der einzufügende Knoten hat eine kleinere Bewertung. Wir landen wieder bei Knoten 3 und stellen fest, daß hier wiederum eine größere Bewertung einzufügen ist, ebenso bei Knoten 4. Das Schema unseres neuen Binär-Baumes finden Sie im Bild 4/4.7.1.2.1-7 und den zugehörigen Diskettenauszug in Bild 4/4.7.1.2.1-8.

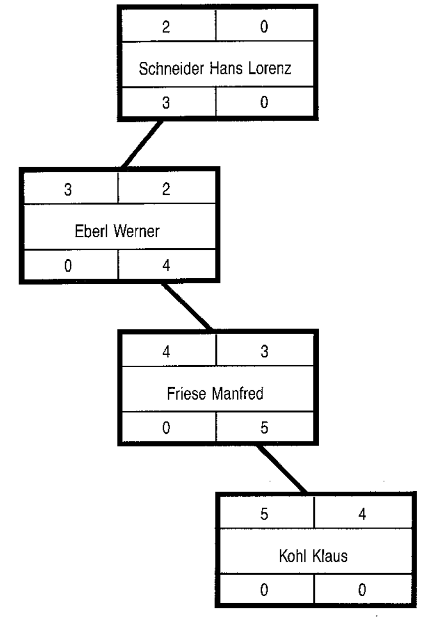


Bild 4/4.7.1.2.1-7 Binär-Baum nach Einfügen von „Kohl Klaus".

Datei: Kunden	Anzahl der Datensätze: 4			
Bewertung	Nummer	Links	Rechts	Vater
Schneider Hans Lorenz	2	3	0	0
Eberl Werner	3	0	4	2
Friese Manfred	4	0	5	3
Kohl Klaus	5	0	0	4

Bild 4/4.7.1.2.1-8 Datei Kundenkey nach Aufnahme von „Kohl Klaus".

Nach dem gleichen Schema fügen wir nun den Eintrag „König Rainer" ein. Die Vorgehensweise sollten Sie anhand von Bild 4/4.7.1.2.1-7 diesmal selbst nachvollziehen. Das Ergebnis finden Sie in den Bildern 4/4.7.1.2.1-9 und 4/4.7.1.2.1-10.

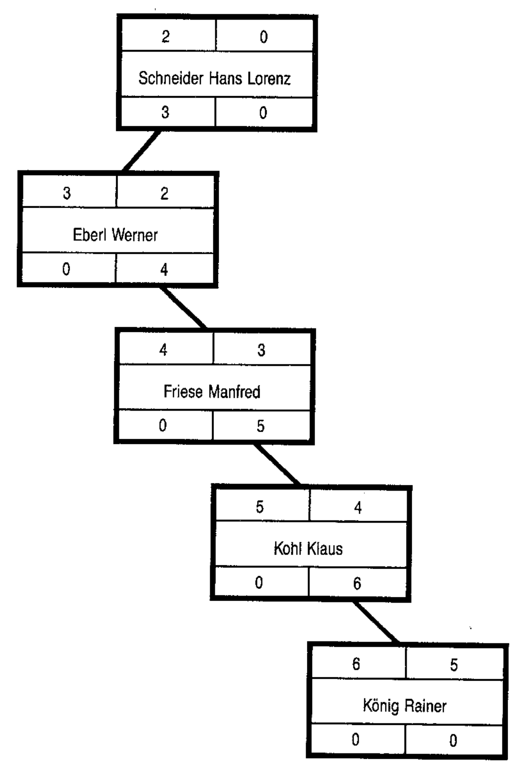


Bild 4/4.7.1.2.1-9 Binär-Baum nach Einfügen von „König Rainer".

Datei: Kunden	Anzahl der Datensätze: 5			
Bewertung	Nummer	Links	Rechts	Vater
Schneider Hans Lorenz	2	3	0	0
Eberl Werner	3	0	4	2
Friese Manfred	4	0	5	3
Kohl Klaus	5	0	6	4
König Rainer	6	0	0	5

Bild 4/4.7.1.2.1-10 Datei Kundenkey nach Aufnahme von „König Rainer".

Zur weiteren Übung sollten Sie noch die Einträge „Roth Thomas" und „Naftanail Adrian" einfügen. Die jeweiligen Ergebnisse finden Sie in den Bildern 4/4.7.1.2.1-11 bis 4/4.7.1.2.1-14.

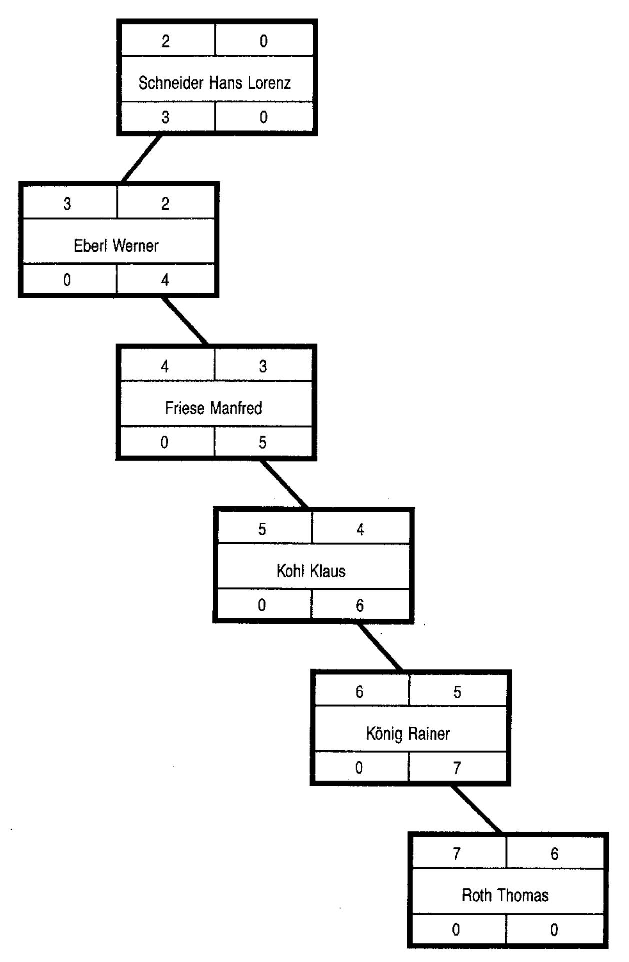


Bild 4/4.7.1.2.1-11 Binär-Baum nach Einfügen von „Roth Thomas".

Datei: Kunden	Anzahl der Datensätze:	6		
Bewertung	Nummer	Links	Rechts	Vater
Schneider Hans Lorenz	2	3	0	0
Eberl Werner	3	0	4	2
Friese Manfred	4	0	5	3
Kohl Klaus	5	0	6	4
König Rainer	6	0	7	5
Roth Thomas	7	0	0	6

Bild 4/4.7.1.2.1-12 Datei Kundenkey nach Aufnahme von „Roth Thomas".

Datei: Kunden	Anzahl der Datensätze:	7		
Bewertung	Nummer	Links	Rechts	Vater
Schneider Hans Lorenz	2	3	0	0
Eberl Werner	3	0	4	2
Friese Manfred	4	0	5	3
Kohl Klaus	5	0	6	4
König Rainer	6	0	7	5
Roth Thomas	7	8	0	6
Naftanail Adrian	8	0	0	7

Bild 4/4.7.1.2.1-14 Datei Kundenkey nach Aufnahme von „Naftanail Adrian".

Wenn Sie sich jetzt den Binär-Baum betrachten, so werden Sie sagen, was soll der Binär-Baum, das gleiche erreiche ich auch, wenn ich die Datendatei sequentiell nach dem gewünschten Schlüssel durchsuche. Stimmt! Ein sequentielles Durchsuchen der Datendatei ist sogar noch schneller, da das Verwalten der Zeiger wegfällt. In beiden Fällen haben wir eine mittlere Zugriffszahl von 4, da sieben Einträge linear durchsucht werden.

Die mittlere Zugriffszahl resultiert aus der Summe der Zugriffe für die einzelnen Datensätze dividiert durch deren Anzahl. Aber bisher hatten wir auch den Spezialfall, daß immer an dem schon bestehenden Teilbaum ein Knoten angefügt wurde.

Ergänzen wir nun unseren Binär-Baum um den Eintrag „Strapko Peter". Bereits bei der Wurzel stellen wir fest, daß dieser Eintrag größer ist und deshalb dort als rechter Sohn angehangen wird. Obwohl wir jetzt den achten Eintrag in die Datei haben, wird dieser Eintrag mit nur zwei Zugriffen gefunden. Bei einer sequentiellen Suche hätten wir jetzt eine mittlere Zugriffszahl von 4,5, durch den Binär-Baum hat sich jedoch unsere vorherige Zugriffszahl auf 3,75 verringert. Den neuen Binär-Baum und den Auszug aus der Binär-Baum-Datei finden Sie in den Bildern 4/4.7.1.2.1-15 und 4/4.7.1.2.1-16.

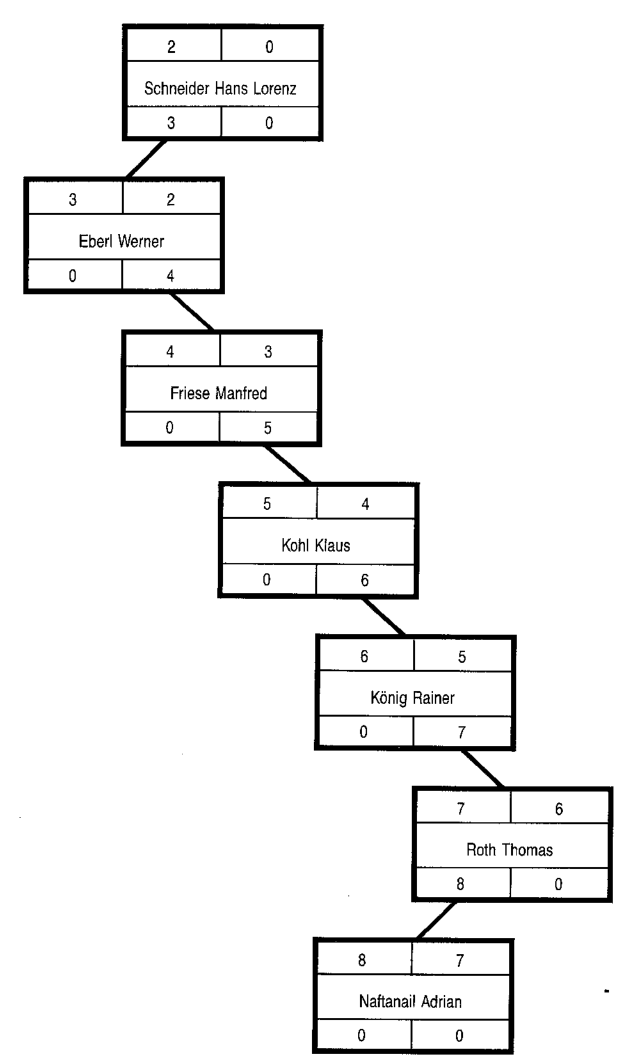


Bild 4/4.7.1.2.1-13 Binär-Baum nach Einfügen von „Naftanail Adrian".

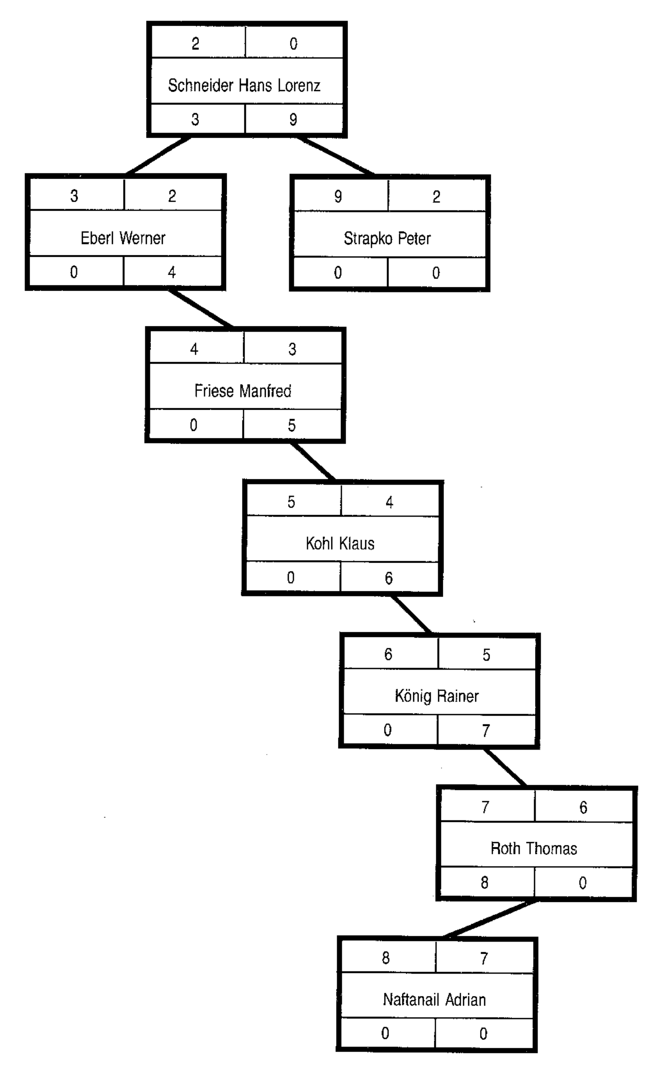


Bild 4/4.7.1.2.1-15 Binär-Baum nach Einfügen von „Strapko Peter".

Datei: Kunden	Anzahl der Datensätze: 8			
Bewertung	Nummer	Links	Rechts	Vater
Schneider Hans Lorenz	2	3	9	0
Eberl Werner	3	0	4	2
Friese Manfred	4	0	5	3
Kohl Klaus	5	0	6	4
König Rainer	6	0	7	5
Roth Thomas	7	8	0	6
Naftanail Adrian	8	0	0	7
Strapko Peter	9	0	0	2

Bild 4/4.7.1.2.1-16 Datei Kundenkey nach Aufnahme von „Strapko Peter".

Als nächstes ergänzen wir den Eintrag „Dietz Sigrid". Die zugehörigen Abbildungen finden Sie in Bild 4/4.7.1.2.1-17 und Bild 4/4.7.1.2.1-18. Die neue Bewertung ist kleiner als die Wurzel, also gehen wir wieder auf den linken Sohn über. Da die neue Bewertung auch kleiner als „Eberl Werner" ist, muß auch hier der linke Sohn in Anspruch genommen werden.

Datei: Kunden	Anzahl der Datensätze: 9			
Bewertung	Nummer	Links	Rechts	Vater
Schneider Hans Lorenz	2	3	9	0
Eberl Werner	3	10	4	2
Friese Manfred	4	0	5	3
Kohl Klaus	5	0	6	4
König Rainer	6	0	7	5
Roth Thomas	7	8	0	6
Naftanail Adrian	8	0	0	7
Strapko Peter	9	0	0	2
Dietz Sigrid	10	0	0	3

Bild 4/4.7.1.2.1-18 Datei Kundenkey nach Aufnahme von „Dietz Sigrid".

Mit den letzten beiden Eintragungen hat also die Ebenenzahl des Binär-Baumes nicht zugenommen. Die mittlere Zugriffszahl würde jetzt bei einer sequentiellen Durchsuchung schon 5 betragen und bei unserem Binär-Baum hat sie weiter auf 3,67 abgenommen. Selbst wenn wir jetzt noch zwei ungünstige Fälle heranziehen und „Strauß Marga" und „Turba Wolfgang" in den Baum einfügen (die entsprechenden Bilder finden Sie unter den Nummern 4/4.7.1.2.1-19 bis 4/4.7.1.2.1-22), so verbessert sich das Verhältnis der mittleren Zugriffszahlen von 6,0 bei der sequentiellen Durchsuchung auf 3,82 bei unserem Suchbaum.

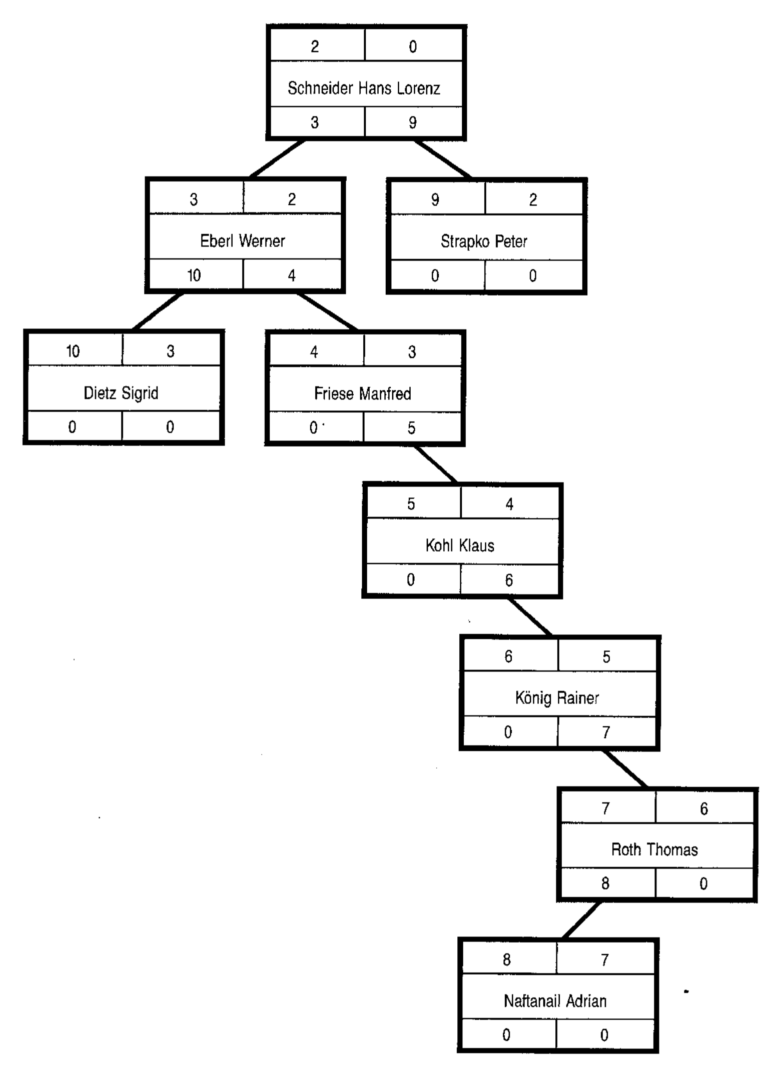


Bild 4/4.7.1.2.1-17 Binär-Baum nach Einfügen von „Dietz Sigrid".

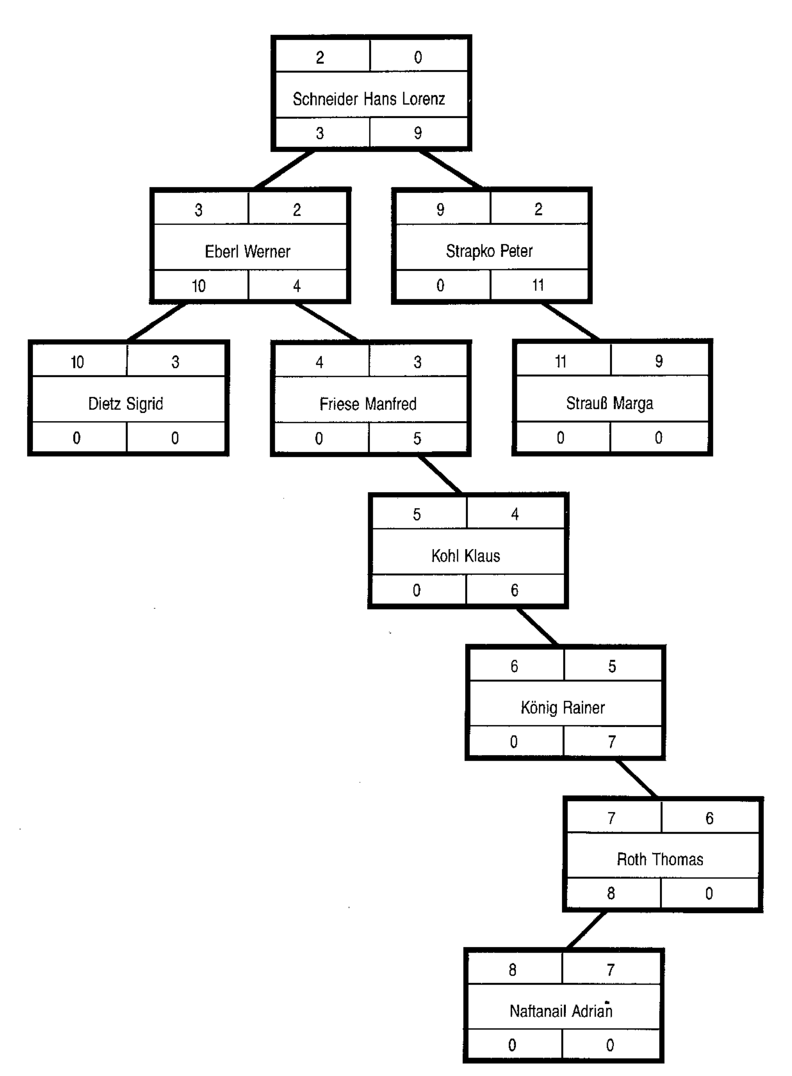


Bild 4/4.7.1.2.1-19 Binär-Baum nach Einfügen von „Strauß Marga".

Datei: Kunden	Anzahl der Datensätze: 10			
Bewertung	Nummer	Links	Rechts	Vater
Schneider Hans Lorenz	2	3	9	0
Eberl Werner	3	10	4	2
Friese Manfred	4	0	5	3
Kohl Klaus	5	0	6	4
König Rainer	6	0	7	5
Roth Thomas	7	8	0	6
Naftanail Adrian	8	0	0	7
Strapko Peter	9	0	11	2
Dietz Sigrid	10	0	0	3
Strauß Marga	11	0	0	9

Bild 4/4.7.1.2.1-20 Datei Kundenkey nach Aufnahme von „Strauß Marga".

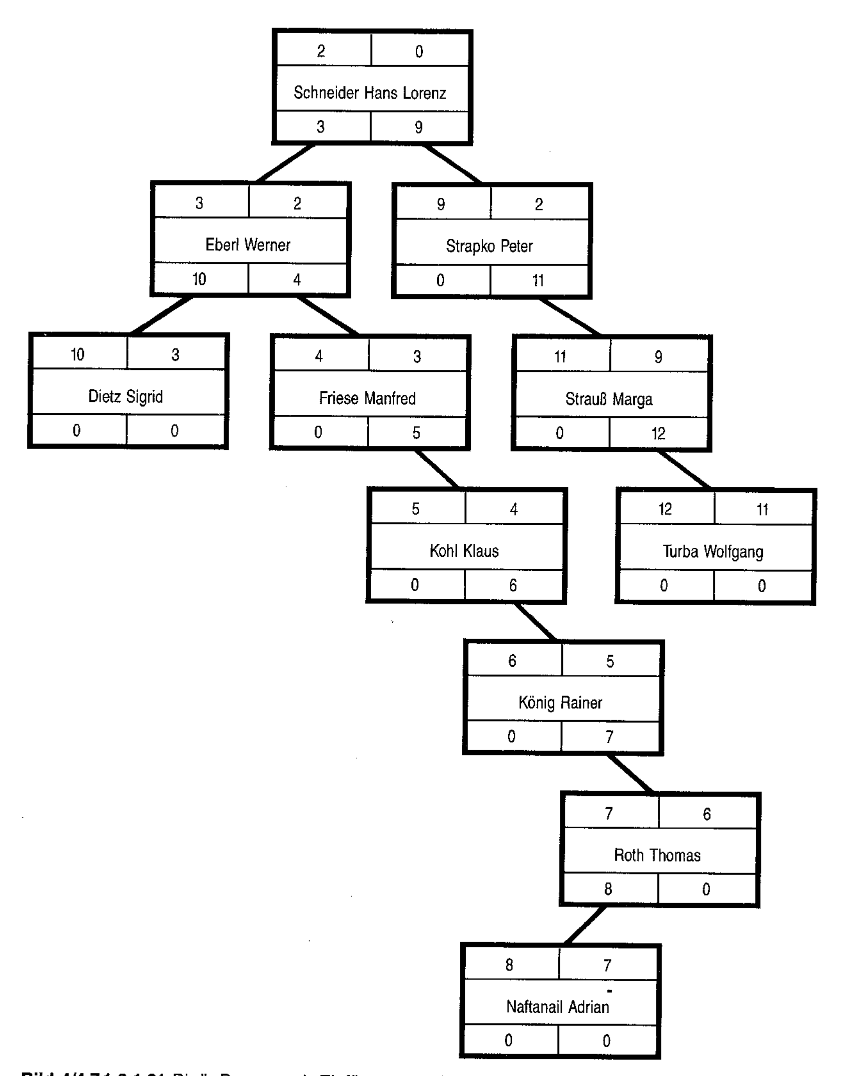


Bild 4/4.7.1.2.1-21 Binär-Baum nach Einfügen von „Turba Wolfgang“.

Datei: Kunden	Anzahl der Datensätze: 11			
Bewertung	Nummer	Links	Rechts	Vater
Schneider Hans Lorenz	2	3	9	0
Eberl Werner	3	10	4	2
Friese Manfred	4	0	5	3
Kohl Klaus	5	0	6	4
König Rainer	6	0	7	5
Roth Thomas	7	8	0	6
Naftanail Adrian	8	0	0	7
Strapko Peter	9	0	11	2
Dietz Sigrid	10	0	0	3
Strauß Marga	11	0	12	9
Turba Wolfgang	12	0	0	11

Bild 4/4.7.1.2.1-22 Datei Kundenkey nach Aufnahme von „Turba Wolfgang“.

In unserem Fall haben wir es mit einem unregelmäßigen Binär-Baum zu tun. Der Aufbau des Binärbaumes ist auch direkt von der Reihenfolge der Eingabe abhängig, wie wir später an einem extremen Beispiel noch aufzeigen wollen.

In unserem Fall haben wir Blätter (Knoten ohne Söhne) in unterschiedlichen Ebenen. Sind alle Blätter auf maximal zwei Ebenen verteilt, so spricht man auch von einem AVL-Baum, der natürlich beim Suchen den Idealfall darstellt.

Wird bei einem AVL-Baum beim Einfügen noch ein Blatt in eine zusätzliche dritte (Blatt-)Ebene befördert, so muß allerdings im AVL-Baum eine Umstrukturierung vorgenommen werden, so daß wieder alle Blätter auf zwei Ebenen verteilt sind. Diese Umstrukturierung ist sehr rechenzeitaufwending und nur dann sinnvoll, wenn aufgrund seltener Eingaben sehr häufige Suchabfragen gefordert sind, was insbesondere bei Informationswiedergewinnungssystemen oder kommerziellen Datenbanken der Fall ist.

In unserem Fall wollen wir uns dieser Mühe nicht unterziehen, da wir aufgrund der Speicherkapazität unserer Floppy-Laufwerke auch keine Bäume mit allzu großen Tiefen (vielen Ebenen) erreichen werden. Trotzdem läßt sich — durch Zufall — auch bei Binär-Bäumen eine solche Anordnung erreichen. Vielleicht kann man dem Zufall auch etwas auf die Sprünge helfen, wenn man seine Daten in geschickter Reihenfolge eingibt. Um entartete Bäume — wie wir zum Schluß noch einen aufzeigen werden — in einen halbwegs vernünftig aufgebauten Binär-Baum umzuwandeln, stellen wir später noch ein Hilfsprogramm vor.

Betrachten wir uns die Zugriffszahlen bei Binär-Bäumen gegenüber einer sequentiellen Durchsuchung anhand des günstigsten Falles. Dies wäre ein AVL-Baum mit Blättern in maximal zwei verschiedenen Ebenen, die natürlich direkt aufeinander folgen. In diesem Falle spricht man auch von einem ausbalancierten Baum. Die folgende kleine Tabelle zeigt Ihnen die maximale Zugriffszahl aufgrund vorgegebener Datensatzanzahl.

Anzahl Datensätze	maximale Zugriffe bei sequentieller Durchsuchung	maximale Zugriffe bei balanciertem Binär-Baum
1	1	1
bis 3	3	2
bis 7	7	3
bis 15	15	4
bis 31	31	5
bis 63	63	6
bis 127	127	7
bis 255	255	8
bis 511	511	9
bis 1024	1024	10
bis 2047	2047	11

Besonders bei großen Bäumen lohnt es sich also, da man — wie im letzten Fall — den Aufwand um fast den Faktor 200 reduzieren kann. Mehr als 2000 Datensätze wird wahrscheinlich die Floppy des C 128 nicht fassen, außer man benutzt ganz kurze Datensätze.

Da man jedoch nicht immer vom Idealfall ausgehen kann, haben Studien ergeben, daß man die mittlere Anzahl von Zugriffen in einem Binär-Baum mit $2 \times \sqrt{n/2}$ angeben kann, wobei n die die Anzahl der Datensätze angibt. Der Teil $2 \times \sqrt{n}$ gibt dabei die Anzahl der Ebenen im Binär-Baum an. In einem Binär-Baum mit 10000 Einträgen sind also unter realistischen Bedingungen im Mittel nur ca. 100 Suchzugriffe auszuführen.

Eigene Überprüfungen an Binär-Bäumen mit 1000 bis 2000 Datensätzen haben ergeben, daß jeweils ein Drittel aller Knoten zwei Söhne aufweist, ein weiteres Drittel nur einen Sohn und das letzte Drittel Blätter sind.

Ein schlechtes Beispiel für einen Binär-Baum hatten wir nach den ersten Einträgen, wo die Anzahl der Knoten gleich der Anzahl der Ebenen war. Versuchen Sie nun mal einen neuen Baum aufzubauen, indem Sie die Einträge in folgender Reihenfolge eingeben:

Dietz Sigrid
Eberl Werner
Friese Manfred
König Rainer
Kohl Klaus
Naftanail Adrian
Roth Thomas
Schneider Hans Lorenz
Strapko Peter
Strauß Margaret
Turba Wolfgang

Sie erhalten dabei einen entarteten Binär-Baum in seiner schlimmsten Form. Das Ergebnis haben wir in Bild 4/4.7.1.2.1-23 dargestellt. Anhand des Beispieles sehen

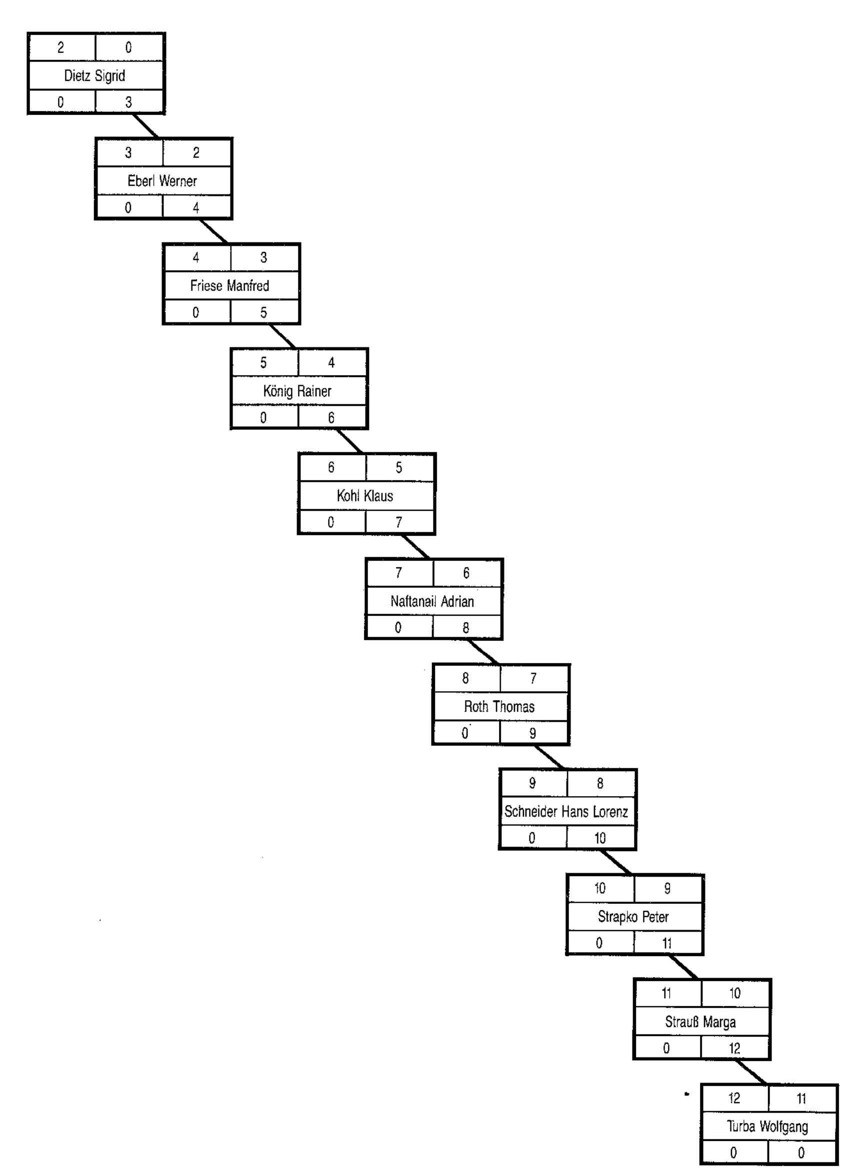


Bild 4/4.7.1.2.1-23 Beispiel für einen entarteten Binär-Baum

Sie schon, daß also bei einer Übernahme von Daten in eine Dateiverwaltung auf Computer, die mit Binär-Baum arbeitet, ein Vorgehen nach einer Kartei oder überhaupt in alphanumerischer Form nicht sehr ratsam ist.

4/4.7.1.2.2

Löschen von Knoten

Wenn ein Datensatz gelöscht wird, muß natürlich auch dessen Knoten aus dem Suchbaum entfernt werden. Dies ist nicht ganz so einfach, wie wir im folgenden noch sehen werden.

Grundsätzlich sind beim Löschen eines Knotens drei Fälle zu unterscheiden:

a) Der zu löschende Knoten hat keinen Sohn
b) Der zu löschende Knoten hat einen Sohn
c) Der zu löschende Knoten hat zwei Söhne

Unser Fall a) ist offensichtlich der einfachste. Das Blatt wird einfach vom Ast „abgepflückt", wobei wir natürlich die „Wunde" verschließen müssen, d.h. den Eintrag auf den entsprechenden Sohn beim Vater auf „0" setzen. In unserem Beispiel-Binär-Baum aus Bild 4/4.7.1.2.1-21 wollen wir dazu den Knoten „Turba Wolfgang" entfernen. Das Ergebnis brauchen wir hier nicht explizit darzustellen, da es aus Bild 4/4.7.1.2.1-19 ersichtlich ist. Dies ist natürlich der einfachste Fall. Bei unserem Fall b) müssen wir schon etwas mehr Arbeit aufwenden.

Wenn ein Knoten nur einen Sohn hat, können wir ihn auch direkt aus dem Binär-Baum entfernen, jedoch müssen wir dessen Sohn bei seinem Vater wieder anfügen. In unserem Beispiel-Binär-Baum (nach der vorangegangenen Löschung) wollen wir den Eintrag „König Rainer" entfernen. Dazu ist beim Knoten mit der Bewertung „Kohl Klaus" als rechter Sohn die Bewertung „Roth Thomas" anzuhängen und entsprechend beim letztgenannten Knoten der Zeiger auf den Vater richtig zu stellen. Die Bilder 4/4.7.1.2.2-1a und 1b stellen diesen Sachverhalt schematisch dar. Beim Vater des zu löschenden Sohnes wird dessen Enkel und neuer Sohn natürlich nicht an einer beliebigen Stelle angenommen, sondern an der, wo der gelöschte Knoten angehangen hat.

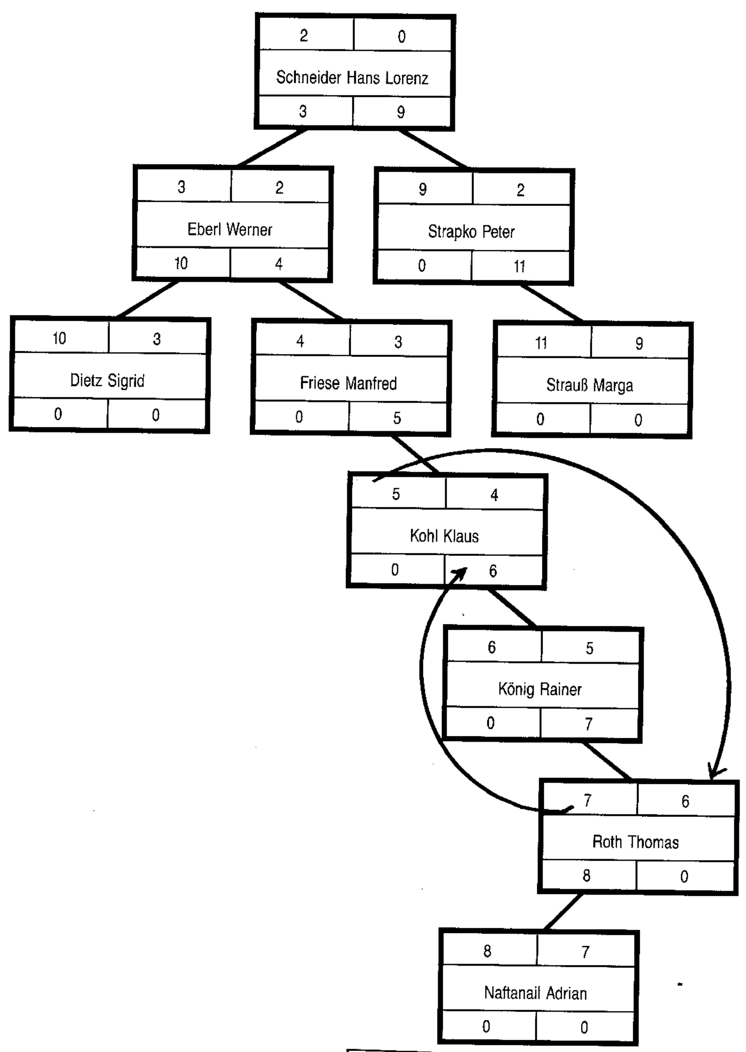


Bild 4/4.7.1.2.2-1 a Löschen eines Knotens (hier „König Rainer") nach Fall b)

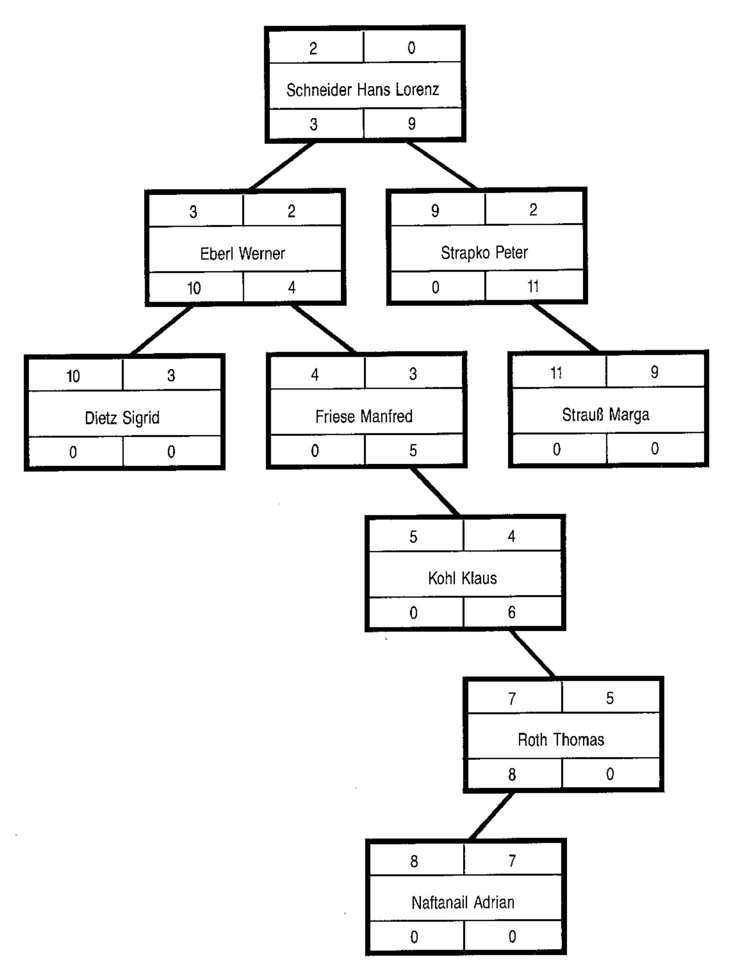


Bild 4/4.7.1.2.2-1 b Binär-Baum nach erfolgter Löschung (siehe 1 a)

An dieser Stelle wollen wir noch eine Maßnahme erläutern, die direkt nichts mit der Handhabung der Binär-Bäume zu tun hat, aber bei unserem Programm verwendet wird: Der gelöschte Knoten und dessen zugehöriger Datensatz belegen natürlich Speicherplatz auf der Diskette, da die Datensatznummer in diesem Falle nicht mehr angesprochen werden kann. Hier ist es sinnvoll, den **letzten** Datensatz in der Datei an die freiwerdende Stelle zu holen, was in unserem Programm die entsprechenden Unterprogramme übernehmen. Natürlich erhält der nach vorne geholte Knoten dadurch eine andere Knotennummer, die entsprechend geändert werden muß. Außerdem müssen bei den beiden Söhnen und dem Vater die Zeiger auf die neue Nummer eingetragen werden.

In unserem Beispiel wird der letzte Datensatz („Turba Wolfgang" wurde schon gelöscht), also „Strauß Marga" als sechster Datensatz (und natürlich auch mit Knotennummer 6) eingesetzt. In Bild 7/4.7.1.2.2-2 haben wir auch diese Änderung durchgeführt. Wohlgemerkt hat diese Änderung nichts direkt mit der Baumbearbeitung, jedoch mit der Speicherplatznutzung und Speicherplatzersparnis zu tun.

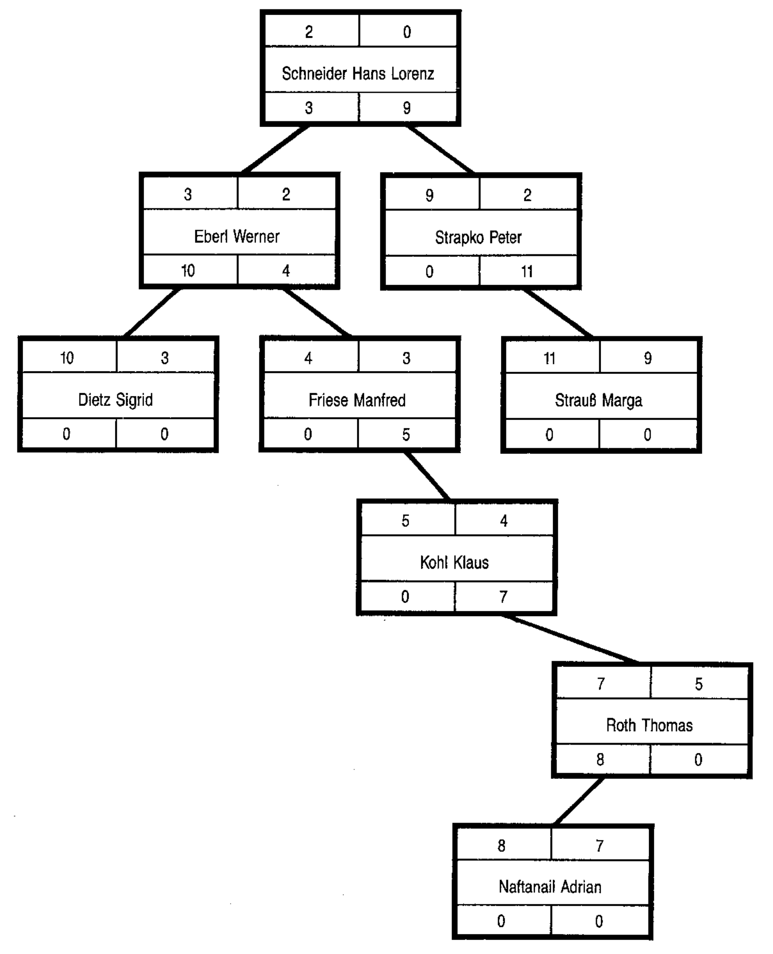


Bild 4/4.7.1.2.2-2 Binär-Baum mit geänderter Nummer bei Knoten „Strauß Marga“

Kommen wir nun zu unserem schwierigsten Fall: c). Wenn ein Knoten gelöscht werden soll und zwei Söhne hat, läßt sich die vorher beschriebene Verfahrensweise natürlich nicht anwenden. Selbst wenn der Vater des zu löschenden Knotens nur den zu löschenden Knoten als Sohn hat, lassen sich beide Söhne des zu löschenden Knotens nicht dort anhängen, da die andere Freistelle nicht genutzt werden kann. Ist z.B. der zu löschende Knoten bei seinem Vater rechts aufgehangen, so kann der linke Sohn des zu löschenden Knotens nicht als linker Sohn des Vaters angehangen werden, da dann unsere Regel (1) verletzt wäre. Naturgemäß sind alle Bewertungen in dem Teilbaum, von dem der zu löschende Knoten die Wurzel darstellt, in diesem Fall größer als die des entsprechenden Vaters. Würde nun der linke Sohn des zu löschenden Knotens als linker Sohn des Vaters verwendet, so wäre der linke Sohn größer, was natürlich nicht sein darf, da er sonst nie mehr gefunden würde.

Die einzige Lösungsmöglichkeit für unseren Fall c) ist es, ihn auf einen der Fälle a) oder b) zurückzuführen. Dazu muß der Knoten mit einem Blatt oder einem Knoten, der nur einen Sohn hat, vertauscht werden. Aber welchen Knoten zieht man dazu heran?

Der auszutauschende Knoten muß auf jeden Fall größer als der linke Sohn des zu löschenden sein und kleiner als dessen rechter Sohn. Es gibt zwei Möglichkeiten, einen Knoten, der beide Bedingungen erfüllt, zu finden: Entweder man nimmt den größten Knoten im Teilbaum des linken Sohnes oder den kleinsten Knoten im Teilbaum des rechten Sohnes, vom zu löschenden Knoten aus gesehen. Auch die Vorgehensweise dazu ist recht einfach: Man geht entweder vom linken Sohn des zu löschenden Knotens aus ausschließlich nach rechts oder vom rechten Sohn des zu löschenden Knotens aus immer nach links.

In unserem Beispiel wollen wir den Knoten „Eberl Werner" löschen. Wenn wir dessen linken Sohn nehmen, so haben wir gleich ein Blatt und brauchen die Suche nicht weiter fortzuführen. Wir können also einfach den Knoten „Dietz Sigrid" gegen den Knoten „Eberl Werner" austauschen. Anschließend könnten wir den Knoten „Eberl Werner" wie unter Fall a) beschrieben abtrennen.

Wir wählen jedoch den anderen Weg und gehen über den rechten Sohn des zu löschenden Knotens. Auch hier wird uns die Sache recht einfach gemacht, da wir bei dem Eintrag „Friese Manfred" im linken Bereich suchen müssen, und hier ebenfalls nichts vorhanden ist. Wir wählen also für unser Beispiel den Tausch „Eberl Werner" gegen „Friese Manfred" und erhalten einen Baum, wie er in Bild 4/4.7.1.2.2-3 dargestellt ist.

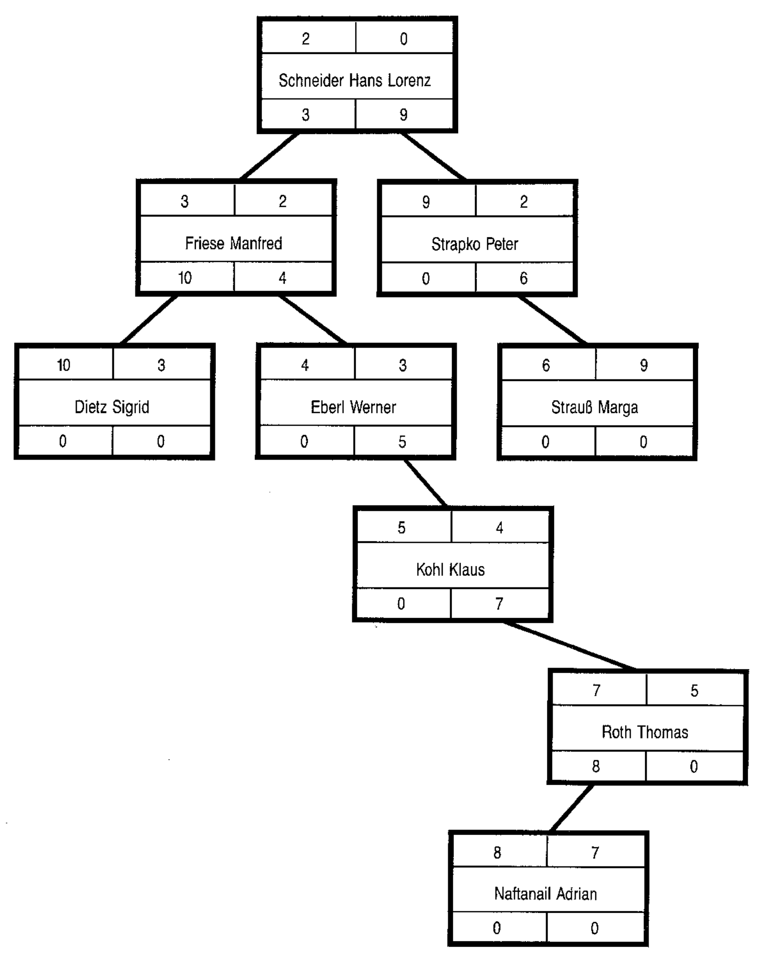


Bild 4/4.7.1.2.2-3 Binär-Baum nach Tausch von „Eberl Werner" gegen „Friese Manfred"

An dieser Stelle sollten Sie sich davon überzeugen, daß der Baum immer noch Regel (1) genügt, wenn wir den Knoten „Eberl Werner" schlußendlich entfernt haben, wie es in Bild 4/4.7.1.2.2-4 der Fall ist.

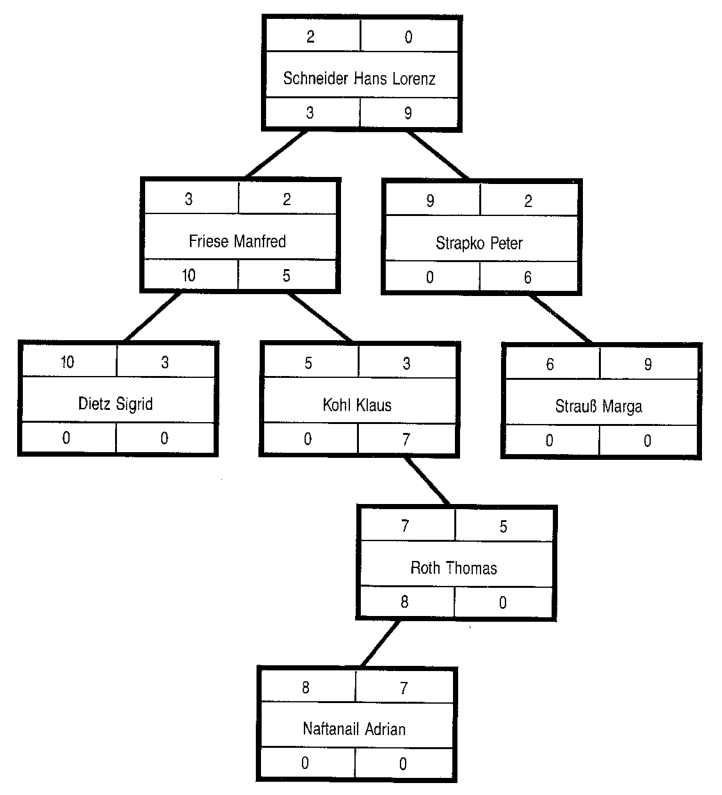


Bild 4/4.7.1.2.2-4 Binär-Baum ohne „Eberl Werner"

Selbst das Löschen der Wurzel ist möglich, wie wir es im folgenden kurz besprechen wollen. Eine Möglichkeit wäre es, im rechten Teilbaum den am weitesten links stehenden Knoten zum Vertauschen heranzuziehen. Da der Eintrag „Strapko Peter" keinen linken Sohn hat, könnte dieser gleich mit der Wurzel vertauscht werden.

Wir wollen Ihnen aber auch zeigen, daß das Verfahren über „größere Strecken" funktioniert und uns für den linken Teilbaum entscheiden. Im linken Teilbaum brauchen wir den Eintrag, der am weitesten rechts steht, also den größten Wert im „kleineren" Teilbaum. Wir gehen also über „Friese Manfred" zu „Kohl Klaus" und „Roth Thomas". Dies ist der am weitesten rechts stehende Knoten, da die Bewertung „Naftanail Adrian" bereits wieder kleiner ist. Wenn wir nun „Schneider Hans

Lorenz“ mit „Roth Thomas“ vertauschen, wie es in Bild 4/4.7.1.2.2-5 dargestellt ist, so ist unsere Ordnung im Baum immer noch erhalten, bis auf den kleinen Schönheitsfehler, daß der Eintrag „Schneider Hans Lorenz“ unserer Regel (1) nicht entspricht.

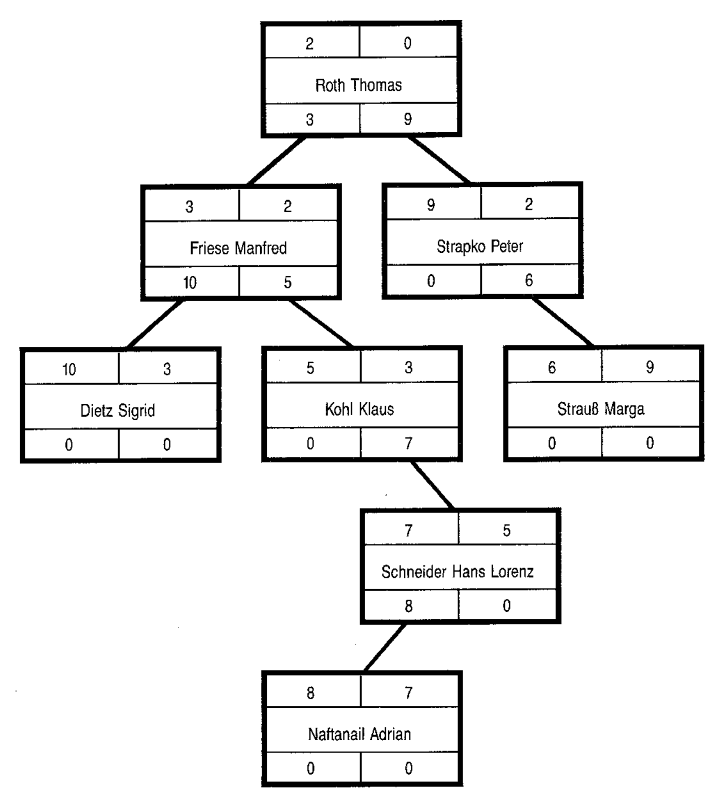


Bild 4/4.7.1.2.2-5 Binär-Baum nach Tausch von „Schneider Hans Lorenz“ gegen „Roth Thomas“

Denn diese Bewertung befindet sich im linken Teilbaum von der Wurzel aus gesehen, wo nur Einträge kleiner als die Wurzel sein dürfen. Aber diesen wollten wir sowieso löschen, woraus sich der Binär-Baum aus Bild 4/4.7.1.2.2-6 ergibt. Auch hier haben wir den Fall c) wieder auf den Fall b) zurückgeführt.

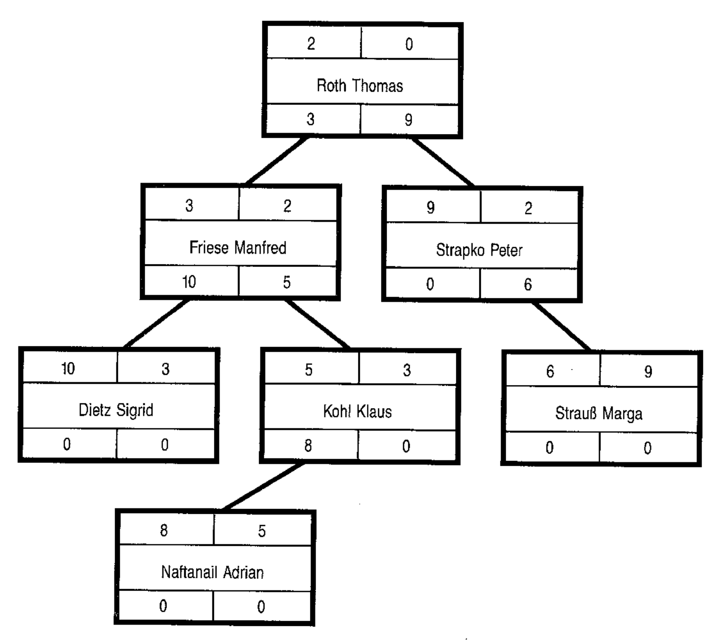


Bild 4/4.7.1.2.2-6 Binär-Baum ohne „Schneider Hans Lorenz"

Umbenennen einer Knotenbewertung

Das Umbenennen einer Knotenbewertung kann mehrere Auswirkungen haben. Einerseits muß sich der Platz des Knotens im Binär-Baum durch die Änderung der Knotenbewertung (gleich Zugriffsschlüssel) nicht unbedingt ändern, andererseits — was am häufigsten der Fall ist — wird eine Neuplazierung im Baum erforderlich. Dies stellt mitunter den kompliziertesten Vorgang bei der Handhabung der Binär-Bäume dar.

Am einfachsten ist noch, wenn man diesen Knoten, gegebenenfalls auch mit Vertauschen, löscht und den Knoten mit der Neubewertung in den Binär-Baum neu einfügt. Bei der Programmierung ist hier sehr große Vorsicht geboten, da einige Knoten an dem „Bäumchen-wechsel-dich-Spielchen" beteiligt sein können.

In diesem Fall muß beim Löschen allerdings berücksichtigt werden, daß hier kein freier Datensatz genutzt werden kann.

4/4.7.1.2.3

Sortierte Drucklisten

Mit dem Suchen und Einfügen sowie Löschen von Knoten sind eigentlich alle wesentlichen Bearbeitungsmöglichkeiten am Binär-Baum aufgezeigt. Der Binär-Baum bietet jedoch noch eine weitere Möglichkeit, nämlich wenn man eine — nach dem Zugriffsschlüssel sortierte — Liste drucken möchte. Ein Sortieren wie in den anderen Fällen wird nämlich dann überflüssig. Übrigens das Sortierprogramm Heap-Short benutzt einen Binär-Baum zum Sortieren. Man braucht nur die Knoten in der alphanumerisch richtigen Weise zu durchlaufen, was durch die genormte Struktur recht einfach ist.

Für das folgende Beispiel wollen wir nicht auf den schon stark dezimierten Knoten aus dem letzten Kapitel zurückgreifen, sondern auf den „voll erblühten" Baum aus Bild 4/4.7.1.2.1-21.

Erkennen Sie schon die Vorgehensweise? Nach unserer Definition sind alle Bewertungen im linken Teilbaum unterhalb der Wurzel kleiner als die Bewertung der Wurzel und im rechten Teilbaum entsprechend größer als die Bewertung der Wurzel. D.h. wir müssen zuerst den linken Teilbaum bearbeiten, dann die Wurzel und dann den rechten Teilbaum.

Betrachten wir uns jetzt den linken Teilbaum. Von diesem der linke Teil ist wieder kleiner als die Wurzel dieses Teilbaumes und diese Wurzel ist wiederum kleiner als alle Einträge im rechten Teilbaum.

Jetzt erkennen Sie sicherlich schon die Periodizität. Sie müssen erst so weit wie möglich links im Baum heruntergehen und finden dort die Bewertung des ersten auszudruckenden Knotens. In unserem Fall ist es der Eintrag „Dietz Sigrid". Als nächstes muß überprüft werden, ob vom gerade ausgegebenen Eintrag ein rechter Teilbaum existiert, der natürlich zuerst bearbeitet werden muß. In unserem Falle liegt dies nicht vor und wir können zum Vater übergehen und drucken die Daten zum Knoten „Eberl Werner".

Da hier ein rechter Teilbaum vorhanden ist, wird dieser nun zur Bearbeitung fällig. Wir lesen die Wurzel dieses Teilbaumes und stellen fest, daß kein linker Sohn vorhanden ist. Also steht die Wurzel dieses Teilbaumes („Friese Manfred") zum Drucken an. In dieser Weise bearbeiten wir auch noch die Bewertungen „Kohl Klaus" und „König Rainer".

Beim Knoten „Roth Thomas" stellen wir fest, daß ein linker Sohn existiert und geben zunächst diesen aus, und dann die Bewertung „Roth Thomas". Nun können

wir uns bis zur Wurzel „hochangeln“, die als nächstes zum Drucken ansteht. Der Rest läßt sich für Sie leicht nachvollziehen.

Generell gehen wir für unser erstes Element von der Wurzel aus soweit nach links — und nie nach rechts — wie möglich. Dieser Knoten wird als erstes bearbeitet, worauf der rechte Teilbaum dieses Knotens folgt. Also immer zuerst links, dann rechts. Der Vater von zwei Teilbäumen wird nach dem linken und vor dem rechten Teilbaum behandelt.

Wenn wir also eine Ebene nach oben steigen, wird der erreichte Knoten verarbeitet, wenn wir von links kommen. Wenn wir von rechts kommen, wird er einfach übergangen und wir gehen weiter nach oben.

Da wir in unseren Listprogrammen nicht nur nach dem Zugrifsschlüssel sortieren wollen, erübrigt sich diese Vorgehensweise in den meisten Fällen. Jedoch haben wir ein solches Unterprogramm in unserer allgemeinen Dateiverwaltung eingebaut, um die Datenstammblätter geordnet ausgeben zu können. Außerdem läßt sich auf diese Weise auch der korrekte Aufbau eines Binär-Baumes prüfen, wie wir in einem Hilfsprogramm noch zeigen werden.

Die Vorgehensweise beim Ausdrucken haben wir in Bild 4/4.7.1.2.3 verdeutlicht.

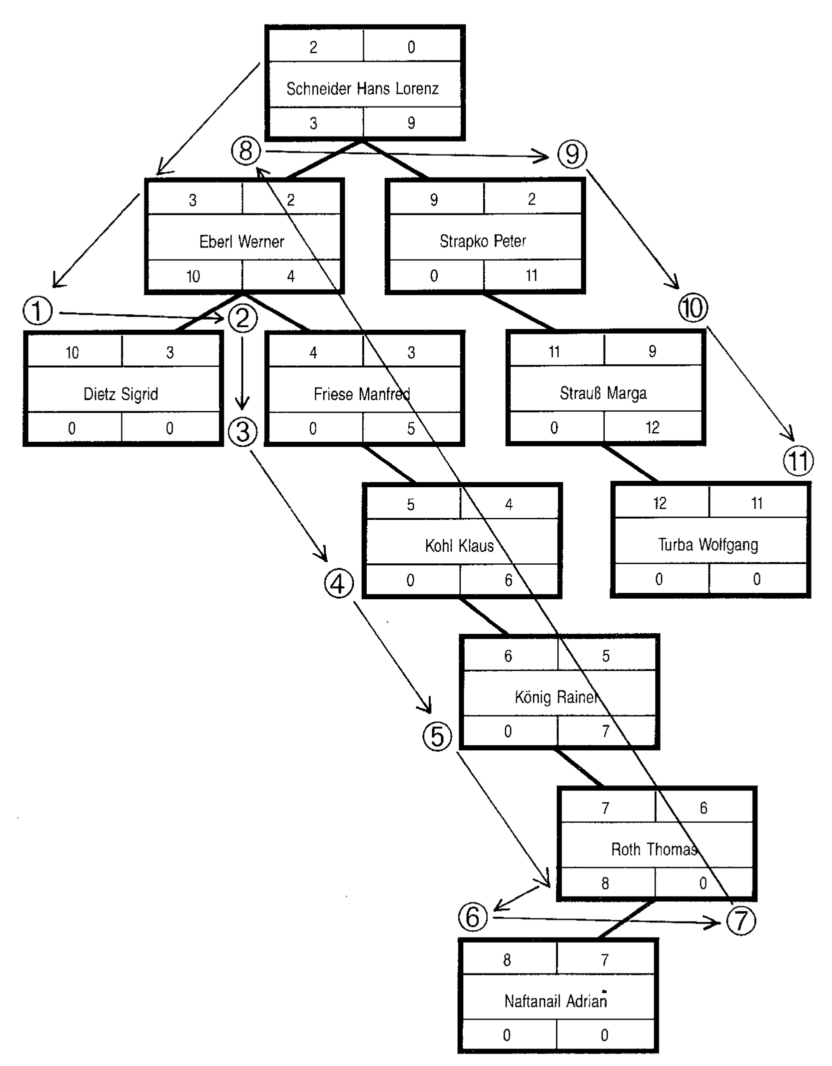


Bild 4/4.7.1.2.3 Schema bei der sortierten Ausgabe von Binär-Bäumen

Die entsprechenden Unterprogramme für die Behandlung der Binär-Bäume finden Sie in unserer Unterprogrammsammlung ab Zeile 20000 (ohne Diskettenzugriff) und ab Zeile 25000 (mit Diskettenzugriff). Beschrieben werden die Unterprogramme in den Kapiteln 4/4.7.2.3 und 4/4.7.2.4.

4/4.7.3

Übersichten

Da der gesamte Komplex der allgemeinen Dateiverwaltung sehr umfangreich und damit auch relativ unübersichtlich ist, wollen wir Ihnen in diesem Kapitel einige Übersichten in die Hand geben, die das Arbeiten mit den Programmen, die Programmierung sowohl als auch die Konzeption dokumentieren. Dabei stellen wir zunächst im ersten Unterkapitel den Zusammenhang zwischen allen Programmen dieser Dateiverwaltung dar. Es folgt eine Übersicht der verwendeten Datendateien und einer Beschreibung aller Variablen, die immer sinngleich in allen Programmen verwendet werden. In diesem Zusammenhang wollen wir von globalen Variablen sprechen.

Als nächstes folgen Zusammenstellungen über die datenstrukturbeschreibenden Variablen, der listenbeschreibenden und etikettenbeschreibenden Variablen. Auch eine Übersicht über alle verwendeten Unterprogramme wird im Rahmen dieses Kapitels gegeben.

4/4.7.3.1

Programmstruktur

An dieser Stelle möchten wir den Zusammenhang zwischen allen in diesem Teil vorgestellten Programmen verdeutlichen. Bild 4/4.7.3.1-1 zeigt diesen Zusammenhang schematisch. Neben den von uns erstellten Programmen können Sie weitere Ihrer Programme sehr einfach in dieses Schema einbinden.

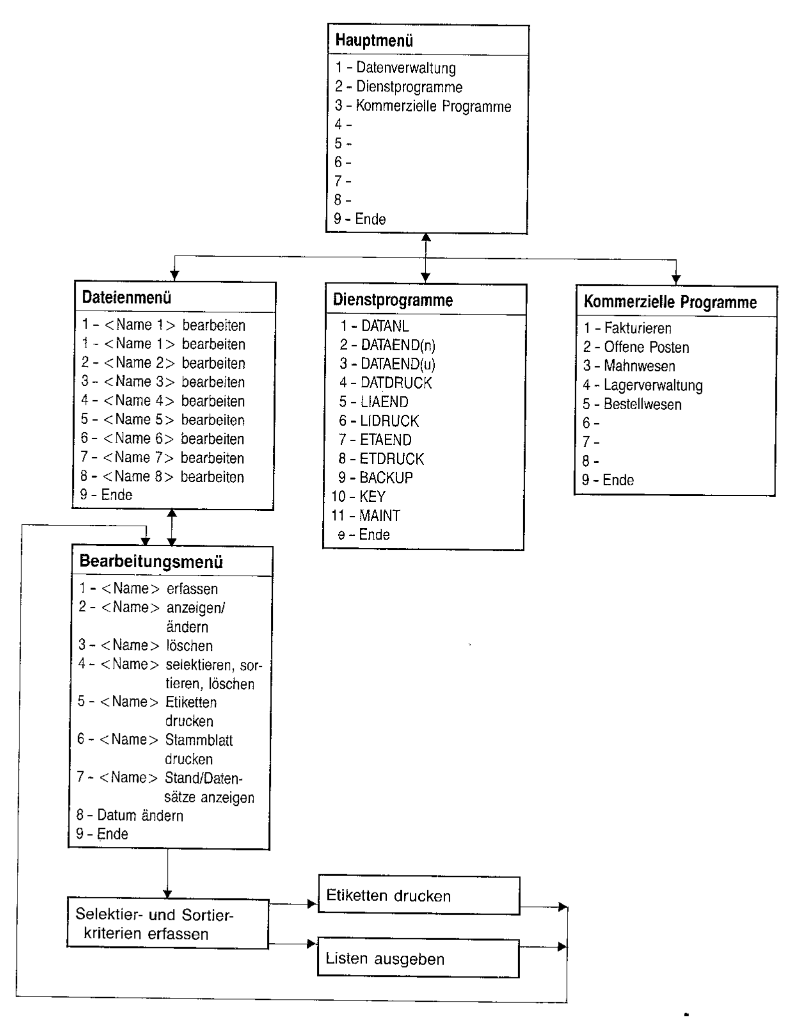

Hauptmenü
1 - Datenverwaltung
2 - Dienstprogramme
3 - Kommerzielle Programme
4 -
5 -
6 -
7 -
8 -
9 - Ende
Dateienmenü
1 - <Name 1> bearbeiten
1 - <Name 1> bearbeiten
2 - <Name 2> bearbeiten
3 - <Name 3> bearbeiten
4 - <Name 4> bearbeiten
5 - <Name 5> bearbeiten
6 - <Name 6> bearbeiten
7 - <Name 7> bearbeiten
8 - <Name 8> bearbeiten
9 - Ende
Dienstprogramme
1 - DATANL
2 - DATAEND(n)
3 - DATAEND(u)
4 - DATDRUCK
5 - LIAEND
6 - LIDRUCK
7 - ETAEND
8 - ETDRUCK
9 - BACKUP
10 - KEY
11 - MAINT
e - Ende
Kommerzielle Programme
1 - Fakturieren
2 - Offene Posten
3 - Mahnwesen
4 - Lagerverwaltung
5 - Bestellwesen
6 -
7 -
8 -
9 - Ende
Bearbeitungsmenü
1 - <Name> erfassen
2 - <Name> anzeigen/ ändern
3 - <Name> löschen
4 - <Name> selektieren, sortieren, löschen
5 - <Name> Etiketten drucken
6 - <Name> Stammblatt drucken
7 - <Name> Stand/Datensätze anzeigen
8 - Datum ändern
9 - Ende
Selektier- und Sortierkriterien erfassen
Etiketten drucken
Listen ausgeben

Bild 4/7.3.1.1

Allem voran stellen wir ein Hauptmenü, das zwischen den drei großen Bereichen Datenverwaltung, Dienstprogramme und kommerzielle Programme verzweigt. Die Menüpunkte 4, 5, 6, 7 und 8 stehen dabei zu Ihrer freien Verfügung. Um möglichst schnell durch die Menüs voranzukommen, werden wir dabei noch einige Tricks verwenden. Mehr dazu in Kapitel 4/4.7.5.

Wenn Sie im Hauptmenü den Punkt 1 (Datenverwaltung) anwählen, so kommen Sie zu den Anwenderprogrammen, mit denen die Daten selbst verwaltet werden können. Dem eigentlichen Bearbeitungsmenü ist ein Dateienmenü vorgeschaltet, mit dem Sie aus den zur Verfügung stehenden Dateien auswählen können. Mit dem Dateienmenü können Sie also zwischen Kunden, Artikeln, Vertretern oder Offene-Posten wählen, oder aber zwischen Schallplattendaten, Briefmarken oder Dias. Dieses Dateienmenü werden wir mit einer Art Automatik ausstatten, so daß Sie die jeweils auf den angelegten Disketten befindlichen Dateien angezeigt bekommen.

Mit dem angeschlossenen Bearbeitungsmenü wählen Sie die Arbeitsschritte zur Datenpflege aus. Das Bearbeitungsmenü ist in das Hauptprogramm der Datenverwaltung integriert und ruft somit für die wichtigsten Bearbeitungsvorgänge keine weiteren Programme auf. Lediglich die Druckroutinen und das Erfassen der dazu benötigten Daten wurde aus dem Hauptprogramm ausgelagert.

Der nächste große Bereich wird durch die Dienstprogramme gebildet. In der Hauptsache handelt es sich in unserem Fall um Programme, die datenbeschreibende Variablen beeinflussen. Es handelt sich hier um Daten, die die **Struktur** der Anwenderdaten beschreiben. Bei den Dienstprogrammen können Sie selbstverständlich eigene Programme ebenso einbinden.

Den letzten Bereich bilden die kommerziellen Programme, d.h. Programme, die auf die verwalteten Daten zurückgreifen. Es handelt sich also um eine hierarchische Struktur, angeführt von vier Menüprogrammen.

4/4.7.3.2

Datendateien und Programmdateien

Neben der Kenntnis der allgemeinen Programmstruktur muß man natürlich auch wissen, mit welchen Dateien man es zu tun hat. Generell müssen wir hier zwischen Datendateien und Programmdateien unterscheiden. Beschäftigen wir uns zunächst mit den Datendateien.

Datendateien			
Typ rel : Relative Datei/Direktzugriffsdatei Typ seq: Sequentielle Datei			
Name	**Typ**	**Inhalt**	**Bemerkung**
Datum	seq	Akt. Tagesdatum	Die sequentielle Datei 'Datum' besteht lediglich aus dem aktuellen Tagesdatum. Sie wird im Programm 'HAUPTMEN' beschrieben und gelöscht.
<Name>da1	rel	Stammdaten (erster Teil)	Gegebenenfalls erster Teil wenn <Name>da2 existiert. Sie wird vom Dateiverwaltungshauptprogramm beschrieben und gelesen, gegebenenfalls werden auch Datensätze gelöscht.
<Name>da2	rel	Zweiter Teil der Stammdaten	Nicht immer vorhanden, was von der vorgegebenen Datenstruktur abhängt. Diese Daten werden ebenfalls vom Dateiverwaltungshauptprogramm gelesen, geschrieben und einzelne Datensätze werden gelöscht.
<Name>dir	seq	strukturbeschreibende Variablen	Diese Datei wird bei der Festlegung der Datenstruktur beschrieben und von den Dateiverwaltungsprogrammen ausgelesen.
<Name>eti	seq	Etikettenformat	Diese Datei wird von den entsprechenden Dienstprogrammen angelegt und geändert und vom Druckprogramm zum Etikettendruck ausgelesen.
<Name>key	rel	Suchbaum	Die Datei <Name>key enthält den Binärbaum, der zum Auffinden der Datensätze aufgrund des in Kapitel 4/4.7.1.2 beschriebenen Verfahrens benötigt wird. (Siehe auch Kapitel 4/4.7.2.5 Binär-Baum-Routinen.)
<Name>li	seq	Listen-beschreibende Variablen	Die Datei wird durch die entsprechenden Dienstprogramme angelegt und geändert und vom Programm zum Drucken der Listen gelesen.

Datendateien			
Typ rel : Relative Datei/Direktzugriffsdatei Typ seq: Sequentielle Datei			
Name	**Typ**	**Inhalt**	**Bemerkung**
<**Name**>**se**	seq	Selektier-kriterien	Vor dem Drucken einer Liste werden die Selektierkriterien erfaßt und in <Name>se abgelegt, wo sie vom Druckprogramm aufgerufen werden.
<**Name**>**so**	seq	Sortierkriterien, siehe Selektierkriterien	
<**Name**>**v1** : <**Name**>**v9**	rel : rel	Stammdaten der ersten verketteten Liste : Stammdaten der neunten verketteten Liste	

Durch <Name> ist jeweils der Name der Dateigruppe gekennzeichnet. Dieser kann vom Anwender frei gewählt werden, sofern er zwölf Zeichen nicht überschreitet. Werden z.B. Kundendaten verwaltet, so existieren auf der Diskette — ohne Berücksichtigung eventuell vorhandener verketteter Listen — folgende Dateien:

- Kundenda1
- Kundenda2 (bei Verwendung einer Doppeldatei)
- Kundendir
- Kundeneti
- Kundenkey
- Kundenli
- Kundense
- Kundenso

Je nach Anzahl der verwendeten verketteten Listen sind auch noch die Dateien Kundenv1 bis Kundenv9 vorhanden.

Werden Dias mit der Dateiverwaltung bearbeitet, so existieren dementsprechend die Dateien:

- Diasda1
- (Diasda2)
- Diasdir
- Diaseti
- Diaskey
- Diasli
- Diasse
- Diasso

Es ist sicherzustellen, daß vor dem Anlegen einer neuen Dateistruktur keine der genannten Dateien auf Diskette existiert.

Nun zu den Programmdateien. Auch diese wollen wir wieder in einer Übersicht zusammenstellen.

Programmdateien	
Menüprogramme	
HAUPTMEN	Hauptmenü, das zu den Bereichen Datenverwaltung Dienstprogramme und kommerzielle Programme verzweigt.
DATEIMEN	Hier werden alle zur Bearbeitung möglichen Dateien dem Anwender automatisch angezeigt.
DIENSTMEN	Menü der Dienstprogramme, insbesondere der Programme zur Verwaltung der datenbeschreibenden Variablen.
SONSTMEN	Menüprogramm für weitere Anwenderprogramme, insbesondere die im Rahmen dieser Dateiverwaltung vorgestellten kommerziellen Programme.
Verwaltungsprogramme	
HAUPTPROGRAMM	der Dateiverwaltung mit den Bereichen Erfassen und Ändern der Daten sowie einiger Nebenfunktionen inklusive dem Bearbeitungsmenü.
SELSORT	Programm zur Erfassung der Selektier- und Sortierkriterien vor einem Listen- bzw. Etikettenausdruck.
LISTDRUCK	Ausdruck der Daten in Listenform, wobei eine gewünschte Listenform noch angewählt werden kann.
ETIDRUCK	Ausdruck von Etiketten.
Dienstprogramme	
DATANL	Programm zum Anlegen der strukturbeschreibenden Variablen
DATAEND	Programm zum Ändern der strukturbeschreibenden Variablen (nur Stammdaten)
DATAENDV	Programm zum Ändern der strukturbeschreibenden Variablen (verkettete Listen)
DATDRUCK	Kontrollausdruck der strukturbeschreibenden Variablen

Programmdateien	
Dienstprogramme	
ETAEND	Programm zum Erfassen und Ändern der etiketten-beschreibenden Variablen
ETDRUCK	Kontrollausdruck der etikettenbeschreibenden Variablen
LIAEND	Programm zum Erfassen und Ändern der listen-beschreibenden Variablen
LIDRUCK	Kontrollausdruck der listenbeschreibenden Variablen
BACKUP	Dienstprogramme zur Datensicherung
KEY	Restaurieren des Binär-Baumes und andere Utilities
MAINT	Programm mit direktem Zugriff auf relative Dateien
Datenverarbeitende Programme	
FAKTUR	Fakturierprogramm
OFFPOST	Offene-Postenverwaltung
MAHN	Mahnwesen
LAGER	Lagerverwaltung
BESTELL	Bestellwesen

Bild 4/4.7.3.2-1 zeigt noch eine Übersicht zwischen den Datendateien und Programmdateien, woraus hervorgeht, welches Programm auf welche Datei schreibend, lesend oder löschend zugreift. Beim Löschen ist noch zwischen dem Löschen der gesamten Datei und dem Löschen eines einzelnen Datensatzes (nur bei relativen Dateien) zu unterscheiden.

In der Tabelle wurden dafür folgende Abkürzungen verwendet:

l — Lesender Zugriff
s — Schreibender Zugriff
r — Löschen eines einzelnen Datensatzes (Record)
d — Löschen der gesamten Datei

Programm	Datum	da1	da2	dir	eti	key	li	se	so	v1 . . . v9
HAUPTMEN	lsd	—	—	—	—	—	—	—	—	—
HAUPTPRG	l	lsr	lsr	l	l	lsr	l	ls	ls	lsd
DATANL	—	*	*	sd*	—	*	—	—	—	*
DATAEND	—	—	—	lsd	—	ls	—	—	—	—
DATAENDV	—	—	—	lsd	—	ls	—	—	—	—
DATDRUCK	—	—	—	l	—	—	—	—	—	—
ETAEND	—	—	—	—	lsd*	—	—	—	—	—
ETDRUCK	—	—	—	—	l	—	—	—	—	—
LIAEND	—	—	—	—	—	—	lsd*	—	—	—
LIDRUCK	—	—	—	—	—	—	l	—	—	—

* = Datei wird angelegt.

Bild 4/4.7.3.2-1: Übersicht über den Zusammenhang zwischen Programmen und Daten (siehe Text)

4/4.7.3.3

Verwendete Variablen von allgemeiner Bedeutung

Es gibt einige Variablen, die ziehen sich durch sehr viele der benötigten Programme oder sie tauchen in einem Programm sehr häufig auf. Bis auf die Hilfsvariablen werden die genannten Variablen ausschließlich mit dem im folgenden aufgeführten Sinn verwendet. Die Übersicht der Variablen von allgemeiner Bedeutung stellt ein weiteres Glied in der Programmdokumentation dar. Aus Gründen des besseren Überblicks werden die Variablen nicht nach Sinngruppen geordnet, sondern alphabetisch aufgeführt.

A$	Temporäre Variable, die nur kurzzeitig verwendet wird, um ein Zeichen von der Tastatur einzulesen.
A0	Da zwei Bildschirmseiten zur Anzeige der einzelnen Datensatzelemente am Bildschirm vorgesehen sind, bildet A0 ein Merker, welche Bildschirmseite gerade aktuell ist.
A1	Anzahl der Datensatzelemente, die in der ersten Bildschirmmaske angezeigt werden. A1 kann Werte 1 und 20 annehmen.
A2	Entsprechend A1 für die zweite Bildschirmseite, gegebenenfalls 0, wenn alle Datenelemente der ersten Bildschirmmaske angezeigt werden.
A6	Da die Datensatzlänge bei den Commodore-Floppys auf 252 Zeichen begrenzt ist, wir aber auch Datensätze mit größerer Länge verwalten wollen, werden die Elemente eines logischen Datensatzes (Datensatz, wie ihn der Anwender sieht) nach Benutzervorgabe in zwei physikalische Dateien aufgeteilt. A6 gibt an, wieviel Datensatzelemente in der ersten Datei (<Name>da1) vorhanden sind.
A7	Siehe A6, jedoch für die zweite Datei, sofern vorhanden (<Name>da2)
AR$	**AR**t eines Datensatzelementes, das gerade im Rechner bearbeitet wird. Insbesondere enthält AR$Werte aus dem Feld AR$(). AR$ kann folgenden Inhalt haben: a:Datensatz enthält alphanumerische Zeichen d:Datensatz enthält Kalenderdatum i :Datensatz enthält ganze Zahlen (Integer) f :Datensatz enthält eine Dezimalzahl, wobei die Anzahl der Vor(x)- und Nachkommastellen (y) in der Form fx.y anzugeben ist.

AR$(AW)	Feld, was als Daten die **AR**t aller Elemente eines Datensatzes enthält (siehe AR$)
AW	**A**nzahl der **W**erte (Datensatzelemente) eines Datensatzes. Gegebenenfalls wird der Wert von AW noch in den Variablen A6 und A7 getrennt mitgeführt (siehe dort)
CD$	Zeichenreihe aus 25x 'Cursor nach unten' (**C**ursor **D**own)
DA$	Tages**DA**tum in der Form TT.MM.JJ. Der Wert von DA$ wird aus der Datei 'Datum' ausgelesen.
DB$	Zeichenkette aus Grafikzeichen für den Strich über der Titelzeile
DU$	Zeichenreihe aus Grafikzeichen zum Unterstreichen der Titelzeile
ET	Anzahl der vorgegebenen **ET**ikettendruckmuster
I,J	Laufvariablen
IH$	Aktueller **I**n**H**alt eines Datensatzelementes
IH$(AW)	Daten eines Datensatzes getrennt nach Datensatzelementen. Der Index stellt eine Ordnung auf die Datensatzelemente dar.
IH%(VL)	Die Zeiger auf die verketteten Listen sind nichts anderes als eine Datensatznummer. Die Zeiger auf das erste Element der verketteten Listen zu einem bestimmten Datensatz wird in diesem Datensatz selbst abgelegt (für den Anwender nicht ersichtlich), und deshalb ebenso Datensatzelemente, wie die Anwenderdaten auch. Physikalisch sind die Zeiger am Ende des Datensatzes angebracht und werden jeweils in das Feld IH%(VL) übernommen.
KL	Länge des Zugriffsschlüssels (**K**ey-**L**ength). Außer dem eigentlichen Zugriffsschlüssel (erstes Element des Datensatzes) werden weitere Daten für die Zeiger benötigt. Siehe dazu auch die Kapitel 4/4.7.1.2 und 4/4.7.2.5
LA	**LA**enge eines Datensatzelementes in Zeichen (zur Erinnerung: Auch Zahlen werden als Zeichenreihen gespeichert)
LA(AW)	Die einzelnen Längen für alle Elemente eines Datensatzes.
LD	**L**änge der ersten **D**atendatei (Länge des Datensatzes), siehe auch A6

LE	Länge der zweiten Datendatei (siehe auch A6)
LW	**L**auf**W**erksnummer unter der die Datendateien (siehe Kapitel 4/4.7.3.2) zu finden sind. Die o.a. Dateien dürfen nicht auf verschiedene Laufwerke verteilt werden.
NA$	**NA**me (Bezeichnung) des aktuellen Datensatzelementes
NA$(AW)	Name (Bezeichnungen) aller Einträge in den Datensatz
NM$	Haupt**N**a**M**e einer Dateigruppe (siehe Kapitel 4/4.7.3.2). NM$ darf die Länge von zwölf Zeichen nicht überschreiten.
SP$	Zeichenreihen aus Leerzeichen (**SP**aces)
ST$	Zeichenreihe aus (Binde)**ST**richen. ST$ wird ausschließlich beim Drucken verwendet.
VI$(VL,5)	In VI$(VL,5) werden die **I**nhalte aller **V**erketteten Listen (jeweils ein Datensatz aus einer verketteten Liste) festgehalten. Der zweite Parameter zeigt auf das entsprechende Element des Datensatzes, und der erste Index gibt die Nummer der verketteten Liste an. VI$(VL,5) ist das Analogon zu IH$(AW).
VL	Anzahl der verketteten Listen

Weitere globale Variablen sind in den folgenden drei Unterkapiteln aufgeführt.

4/4.7.3.4

Datenstrukturbeschreibende Variablen

Die in diesem und den folgenden beiden Unterkapiteln vorgestellten Variablen haben wir — bis auf die wichtigsten — bewußt aus der Aufzählung der Variablen mit allgemeiner Bedeutung herausgehalten, da ihre Wirkung besser zu verdeutlichen ist, wenn sie im Zusammenhang dargestellt sind. Aus diesem Grunde ist der Zusammenhang auch grafisch dargestellt. Die Darstellung für die datenstrukturbeschreibenden Variablen finden Sie in Bild 4/4.7.3.4-1.

Zum Konzept der datenstrukturbeschreibenden Variablen sei auf Kapitel 4/4.7.1.1 verwiesen. Generell benötigen wir die strukturbeschreibenden Variablen sowohl für den Stammdatensatz als auch für jede der verketteten Listen, die ja jede für sich auch eine eigene Datei darstellen. Als wichtigstes Element ist hier die Anzahl der Werte (Elemente) jeden Datensatzes zu nennen. Weiterhin benötigen wir einen Namen für jeden Eintrag sowie dessen Typ und Länge. Im Stammdatensatz müssen weiterhin Zeiger auf die ersten Einträge der zugehörigen verketteten Listen vorhanden sein, und jeder dieser Listen wollen wir auch noch einen eigenen Namen geben. Daraus resultieren die in der folgenden Aufsicht angeführten Variablen:

AR$(AW)	In AR$(AW) sind die Typen der einzelnen Datensatzelemente festgehalten (zur näheren Beschreibung des Inhalts von AR$(AW) siehe Kapitel 4/4.7.3.3).
AW	Anzahl der Elemente eines Datensatzes. Damit ist der gesamte logische Datensatz gemeint, wie er vom Anwender verwendet wird. Die Trennung in zwei physikalische Dateien wird berücksichtigt, indem in AW die Anzahl der Elemente in der ersten Datendatei (<Name>da1) als Nachkommastellen zugeordnet sind. Die Nachkommastellen werden später mit 100 multipliziert und der Variablen A6 (siehe Kapitel 4/4.7.3.3) zugewiesen. A7 errechnet sich dann aus der Differenz zwischen AW und A6. Ist der Wert von AW eine Integerzahl, so wird nur auf eine physikalische Datei zurückgegriffen. Einige Beispiele: AW = 27,14 : Ein Datensatz enthält insgesamt 27 Elemente, davon 14 in der ersten physikalischen Datei AW = 12 : Es wird nur eine physikalische Datei benötigt, die alle zwölf Datensatzelemente enthält AW = 14,7 : Zwei physikalische Datendateien, wo beide je 7 Datensatzelemente enthalten
LA(AW)	erfaßt die Längen der einzelnen Datensatzelemente, wobei zu berücksichtigen ist, daß alle Einträge als Zeichenreihe zu speichern sind. Für maximal sechsstellige Zahlen beträgt die Länge sieben, da das Vorzeichen mitberücksichtigt werden muß und — sofern es sich um Dezimalzahlen handelt, ist zur Anzahl der Vor- und Nachkommastellen noch eine 2 zu addieren (Vorzeichen und Dezimalpunkt)
NA$(AW)	Enthält die Beschreibungen aller Elemente eines Datensatzes. Aus Gründen des Bildschirmaufbaus darf die Länge der einzelnen Einträge von NA$(AW) zwanzig nicht übersteigen.
VA(VL,5)	Entsprechend der Variablen LA(AW) gelten die Inhalte von VA(VL,5) für die Länge der einzelnen Datenelemente in den verketteten Listen.

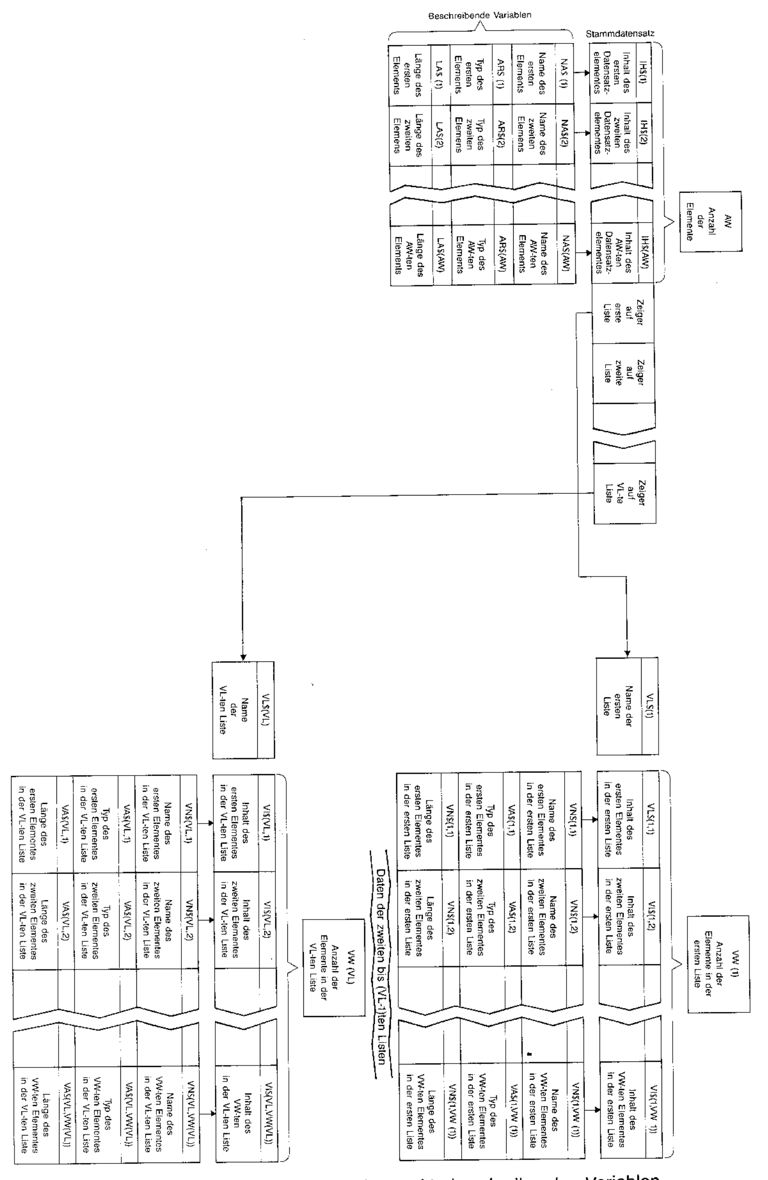


Bild 4/4.7.3.4-1 Schematische Darstellung der strukturbeschreibenden Variablen

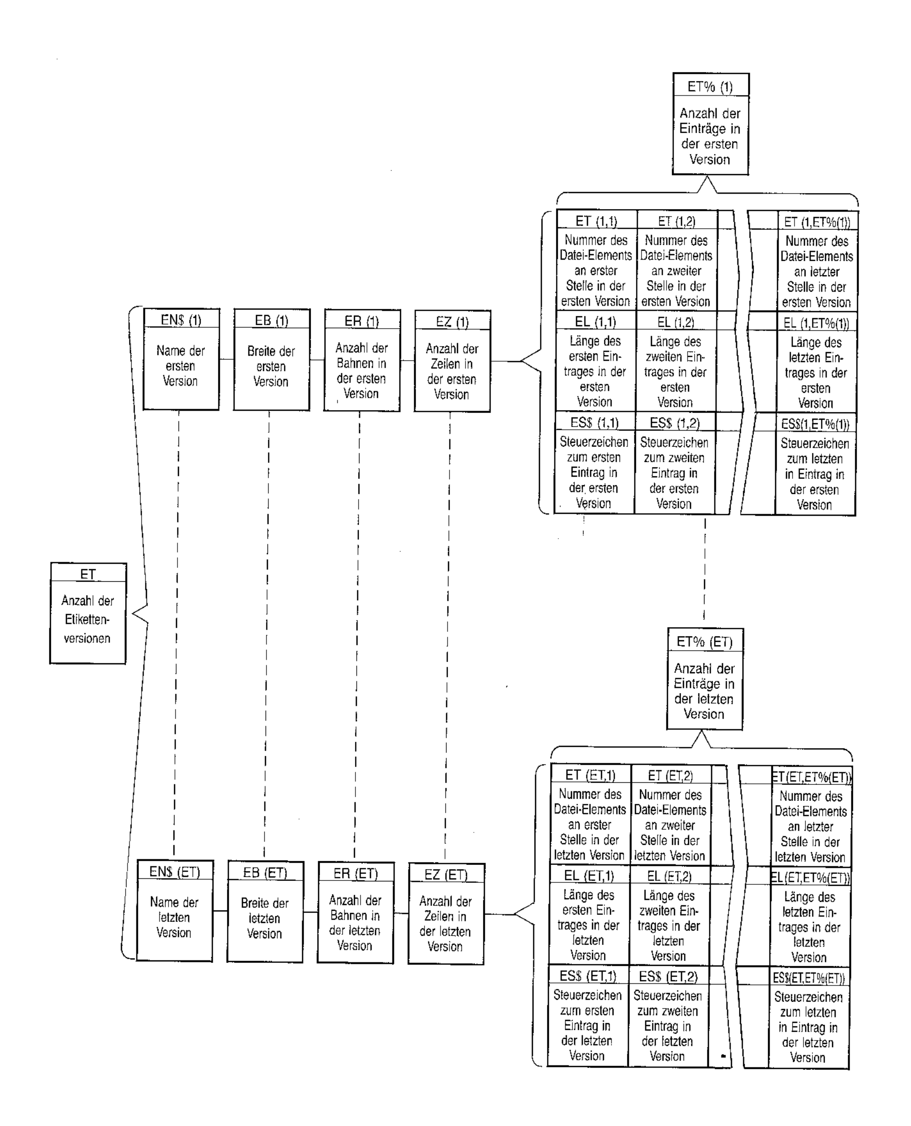


Bild 4/4.7.3.6: Schema der Etikettenbeschreibenden Variablen

VA$(VL,5)	Das Pendant zu AR$(AW) in der Standarddatei bildet VA$(VL,5) bei den verketteten Listen. Der erste Index gibt die Nummer der verketteten Liste an und der zweite Eintrag die Nummer des Datensatzelementes.
VL	Gibt die Anzahl der konfigurierten verketteten Listen an. Beachten Sie bitte, daß für jede verkettete Liste im Standarddatensatz Platz freigehalten werden muß für einen Zeiger auf das erste Element der verketteten Liste. Im Prinzip ist die Anzahl der verketteten Listen nur durch die größtmögliche Datensatzlänge (und das bei zwei physikalischen Dateien!) begrenzt, jedoch haben wir softwaremäßig ihre Zahl auf neun beschränkt. Außerdem lassen wir je verkettete Liste nur maximal fünf Datensatzelemente zu, was wohl in den allermeisten Fällen ausreichen wird. Generell ist es auch möglich, an eine verkettete Liste weitere verkettete Listen anzuhängen, jedoch wollen wir innerhalb unserer Dateiverwaltung davon absehen, da es sich hierbei schon um sehr spezielle Anwendungsgebiete handelt.
VL$(VL)	Speichert die Namen der einzelnen verketteten Listen.
VN$(VL,5)	Ist das Analogon zu NA$(AW) für die verketteten Listen. Auch hier bezeichnet der erste Index wieder die Nummer der verketteten Liste und der zweite Index die Nummer des Eintrages.
VW(VL)	In AW hatten wir die Anzahl der Datensätze in der Stammdatei festgehalten. Dies müssen wir auch für jede einzelne verkettete Liste durchführen, was für uns die Variable VW(VL) erledigt.

Generell haben wir für verkettete Listen Variablennamen gewählt, die mit einem V beginnen und deren zweiter Buchstabe korrespondiert zu dem — meistens auch zweiten Buchstaben — des entsprechenden Variablennamens für die Stammdatei.

4/4.7.3.5

Listenbeschreibende Variablen

Genauso, wie wir bei der Dateiverwaltung vorgehen, werden wir es auch bei einem Teilbereich, den gedruckten Listen, tun. Auch hier soll der Benutzer der Dateiverwaltung selbst festlegen können, wie seine Listen aussehen sollen, und welche Angaben sie enthalten. Dies zwar nur in eingeschränktem Umfang, aber doch so, daß sich alle Informationen auf fast beliebige Art und Weise zusammenstellen lassen. Im folgenden eine Aufzählung der wichtigsten Variablen aus diesem Bereich, deren Zusammenhang aus Bild 4/4.7.3.5-1 ersichtlich ist.

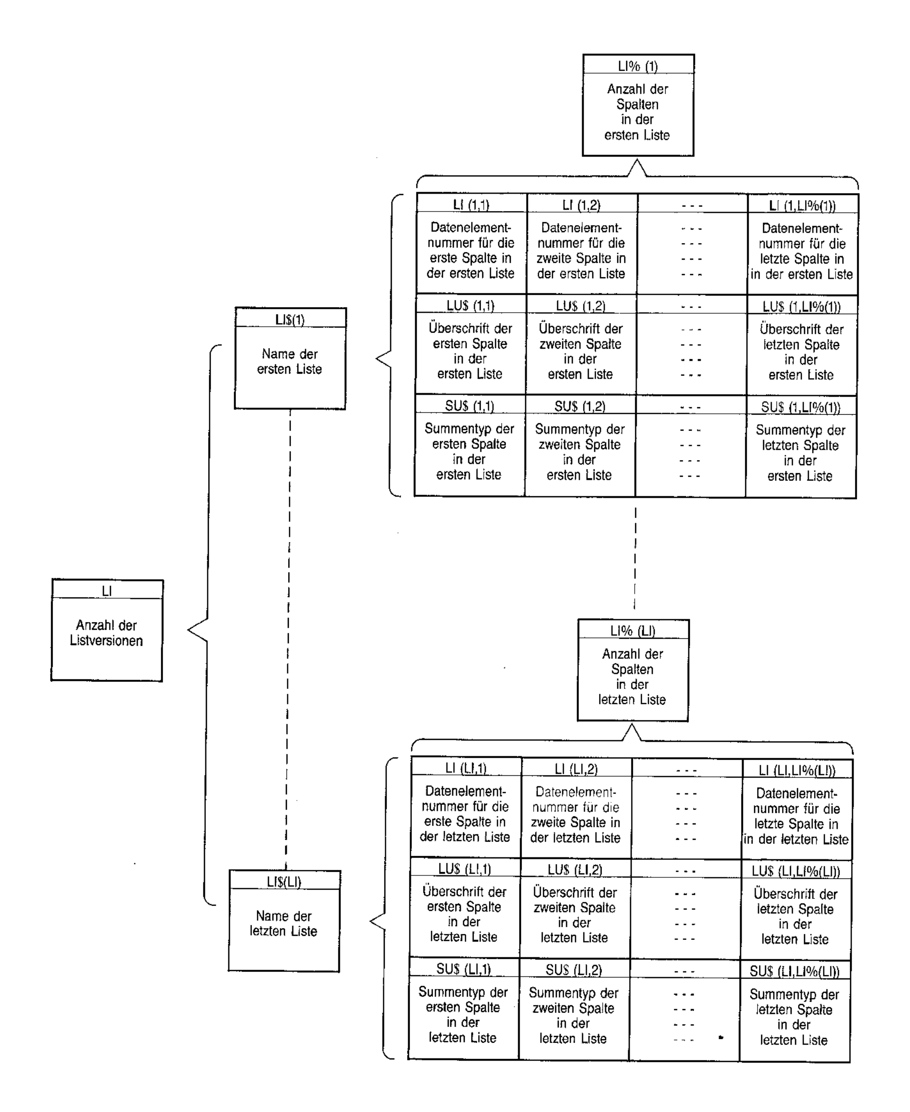


Bild 4/4.7.3.5-1: Schematische Darstellung der listenbeschreibenden Variablen

Die listenbeschreibenden Variablen werden in der Datei ⟨Name⟩li abgelegt.

LI	Anzahl der Listversionen, die prinzipiell nicht begrenzt ist.
LI(LI,AW)	Bei den ausgedruckten Listen soll jeder Anwender selbst entscheiden können, welche Datensatzelemente in welcher Reihenfolge gedruckt werden. Der Listenausdruck ist naturgemäß spaltenweise, wobei in jeder Spalte der Inhalt eines Datenelementes aufgezeigt wird. Für jede Liste muß also gespeichert werden, welches Datensatzelement in welcher Reihenfolge ausgedruckt werden soll. Dafür steht uns der zweite Index in LI(LI,AW) zur Verfügung. Da im ungünstigsten Fall alle Datensatzelemente ausgedruckt werden sollen, muß die Matrix LI(,) im zweiten Index entsprechend groß (AW) dimensioniert werden. Der erste Index gibt die Nummer der Liste an (siehe LI).
LI$(LI)	Für eine anwendergerechte Bedienerführung ist es unerläßlich, daß die Listen nicht nur eine Nummer sondern auch Namen erhalten, mit denen man sie unterscheiden kann. Diese Namen werden in LI$(LI) abgespeichert.
LI%(LI)	Durch LI(LI,AW) ist noch nicht festgelegt, wieviel Elemente die niedrigere Liste letztendlich enthält. Diese Zahl wird in LI%(LI) gespeichert.
LU$(LI,AW)	Generell wollen wir die Bezeichnungen der einzelnen Datensatzelemente zur Listenüberschrift heranziehen, jedoch dem Anwender abweichende Überschriften ermöglichen. Für jede der verschiedenen Listen (erster Index) werden die Überschriften zu den einzelnen Spalten (zweiter Index) in LU$(LI,AW) abgelegt.
SU$(LI,AW)	Natürlich gibt es viele Anwendungsfälle, bei denen nicht nur einfach die Informationen der einzelnen Datensätze ausgedruckt werden sollen, sondern diese auch aufzusummieren sind. SU$(LI,AW) gibt einenen Hinweis auf die Summationsart. Entweder werden Integerzahlen (I) oder Fließkommazahlen (F) oder gar nicht summiert. Im letzten Fall enthält der zuständige Eintrag in der Matrix SU$(,) eine leere Zeichenreihe. Für jede Liste (erster Index) und jede Spalte in der Liste (zweiter Index) wird die Art der Summation festgehalten.

4/4.7.3.6

Etikettenbeschreibende Variablen

Neben dem Ausdruck normaler Computerlisten wollen wir dem Benutzer unserer allgemeinen Dateiverwaltung auch die Möglichkeit in die Hand geben, Etiketten nach seinen Wünschen zu drucken. Dabei greifen wir wieder auf unser bewährtes Schema zurück und lassen den Anwender die Struktur seiner Etiketten selbst bestimmen. In diesem Kapitel stellen wir die wichtigsten dazu benötigten Variablen vor.

Der Druck von Etiketten innerhalb unseres Programmsystems ist nicht nur auf das Anschreiben von Kunden beschränkt, sondern hängt hauptsächlich von der Art und Größe der verwendeten Etiketten beim Anwender ab. Artikelbeschriftungen, Preisauszeichnungen, Übersichten über Tonband- und Datenkassetten, Tonbänder und Schallplatten und viele andere Bereiche lassen sich mit dem Programm zum Ausdruck von Etiketten sehr leicht erledigen, da die Daten ja schon in unserer Dateiverwaltung vorhanden sind. Die Inhalte der nachfolgenden Variablen finden Sie in der Datei ⟨Name⟩ etc.

EB(ET)	Breite der Etiketten in Zeichen gemessen
EL(ET,AW)	Länge eines einzelnen Eintrages beim Druck auf ein Etikett. Im ersten Index wird die Nummer des Etikettenformates und im zweiten Index die Nummer des Datensatzelementes angegeben. Eventuell sind die Daten noch auf das vorgegebene Maß abzuschneiden oder mit Leerzeichen auf die gewünschte Länge aufzufüllen.
EN$(ET)	Wie bei unseren verschiedenen Listenversionen auch, sollen die Etikettenversionen jeweils einen eigenen Namen erhalten, der in EN$() gespeichert ist.
ER(ET)	Im Privatbereich werden sicherlich nur einbahnige Etiketten verwendet, im kommerziellen Bereich sind jedoch sehr häufig mehrbahnige Etikettenbögen in Verwendung. Liegen z.B. vier gleichartige Etiketten nebeneinander, so spricht man von vierbahnig. Die Anzahl der Bahnen für jede einzelne Etikettenversion wird in vorgenannter Variable festgehalten. Die Speicherung der Anzahl der Bahnen ist besonders wichtig, da bei mehreren Schreibzeilen pro Etikett, mit den normalen Druckern nicht zuerst ein Etikett vollständig gedruckt werden kann, sondern die ersten Zeilen aller nebeneinander liegenden Etiketten gleichzeitig auszudrucken sind.
ES$(ET,AW)	Die meisten Matrixdrucker lassen verschiedene Schriftarten (Unterstreichen, Fettschrift, Breitschrift), teilweise sogar verschiedene Zeichensätze zu. In der Regel wird das Umschalten zwischen verschiedenen Schriftarten / Zeichensätzen durch ein Steuerzeichen dem Drucker übermittelt. Um die Eigenschaft der Drucker nutzen zu können, können nach jedem Eintrag im Etikett Steuerzeichen angegeben werden. Diese werden in ES$(,) abgelegt. Im Druckprogramm für Etiketten werden wir später noch eigene 'Steuerzeichen' kreieren, die die an den Drucker zu übergebenden Daten innerhalb des Rechners noch umformen.
ET	Anzahl der Etikettenversionen. Analogon zu LI bei den Listversionen und VL für die verketteten Listen. ET muß als erstes bekannt sein, um die anderen Variablen in diesem Zusammenhang entsprechend dimensionieren zu können.
ET(ET,AW)	Dieses zweidimensionale Feld entspricht der Variablen LI(LI,AW) bei den Listversionen. Hier werden für alle Etikettenversionen (erster Index) die Datensatzelemente (zweiter Index) in der Reihenfolge ihres Erscheinens auf dem Etikett eingetragen.
ET%(ET)	Anzahl der jeweiligen Einträge bei der im Index angegebenen Etikettenversion.
EZ(ET)	Damit Etiketten der unterschiedlichsten Höhen verwendet werden können, wird in EZ(ET) die Anzahl der Schreibzeilen je Etikett festgehalten. Die Ränder am oberen und unteren Rand der Etiketten werden dabei mitgerechnet.

Eine grafische Übersicht der etikettenbeschreibenden Variablen finden Sie in Bild 4/4.7.3.6.

4/5 Maschinensprache

4/5.2 Assemblerkurs

In diesem Kapitel werden Grundlagen und Befehlsstruktur der Assemblerprogrammierung auf dem Commodore 64 vorgestellt. Die Assemblersprache ist die Muttersprache eines Computers, also die Sprache, die direkt vom Computer ausgeführt werden kann.

4/5.2.1 Grundlagen der Assemblerprogrammierung

Alle Programme, die auf Ihrem Computer ausgeführt werden, werden von der CPU abgehandelt. So ist der BASIC-Interpreter des C64 auch nur ein Programm, wie zum Beispiel eine Textverarbeitung. Wenn sie also einen Basic-Befehl eingeben, so wird durch diesen Befehl ein kleines Programm gestartet. Eine CPU versteht nämlich keine Sprachen wie BASIC, PASCAL, FORTH oder andere, sondern nur eine speziell für die CPU festgelegte Folge von Bitmustern. Daraus ergibt sich auch, warum Assemblerprogramme (= Programme die in der Sprache der CPU geschrieben sind) um vieles schneller sind als Basic-Programme: die CPU kann das Programm direkt abarbeiten und muß nicht erst einen Befehl erkennen, das zugehörige Programm suchen und schließlich abarbeiten (d.h. einen Befehl interpretieren). Daher lohnt es sich, die Sprache der CPU zu lernen.

Bevor man sich jedoch mit der Sprache der CPU beschäftigen kann, muß man sich erst einmal ansehen, aus welchen Komponenten die CPU 6510 besteht.

4/5.2.1.1

Die CPU 6510

Das Herz des Commodore 64 ist die CPU 6510 (CPU= Zentraleinheit/engl.: central processing unit). Dieser Baustein steuert alle Abläufe im Computer. Die CPU 6510 ist eine leicht modifizierte Variante der CPU 6502, welche neben der Z-80 eine der weitverbreitetsten Zentraleinheiten ist. Man findet sie in den Computern von Commodore, Atari, Apple und anderen. Die 6510 entspricht in ihrem Befehlssatz genau der 6502. Die CPU 8502 des C 128 ist ebenfalls nur eine modifizierte 6502. Daher gelten die Aussagen über die 6510 auch für die 8502 des C 128. Der C 128 hat jedoch zusätzlich eine Z-80 CPU eingebaut, die im CP/M Mode in Funktion gesetzt wird. Im 64'er Modus und im 128'er Modus arbeitet er mit der 8502.

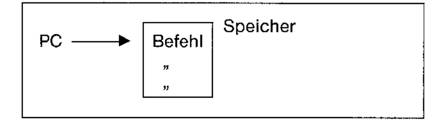
4/5.2.1.2

Das Registermodell der CPU 6510

Die CPU 6510 besteht aus fünf 8-Bit-Registern und einem 16-Bit-Register.

Prozessorstatusregister	(PSR)
Akkumulator	(Akku)
X-Register	(X)
Y-Register	(Y)
Stackpointer	(SP)
Adresszähler (16 bit)	(PC)

Zunächst enthält die CPU einen Befehlszähler PC, der die Adresse des nächsten zu holenden Befehles enthält. Er ist 16-Bit breit d.h. man kann mit ihm 2^{16}=65536 (=64K) verschiedene Adressen ansprechen.



Dann gibt es ein 8-Bit breites Rechenregister, den Akkumulator (Akku). Über den Akkumulator werden alle arithmetischen und logischen Befehle ausgeführt. Mit dem Akkumulator lassen sich die Zahlen 0–255 oder –128 bis +127 darstellen (siehe auch Kapitel über Zahlendarstellung).

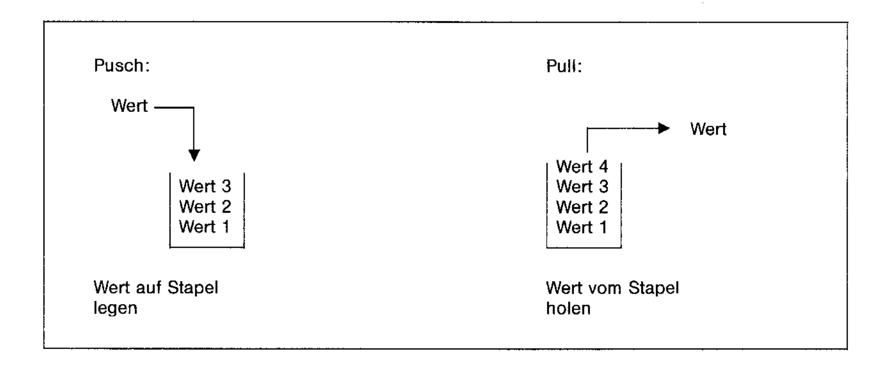
Zwei weitere Register, X und Y genannt, sind besonders für die Adressierungsarten interessant. Mit ihnen werden Tabellen verwaltet, wobei die Register als Offset zum aktuellen Element benutzt werden. Dies bedeutet, daß das Register die Tabelleneinträge zählt, und sich die Adresse des Tabelleneintrages aus der Adresse des Tabellenanfangs addiert mit dem Inhalt des Registers ergibt. Daher werden X und Y Indexregister genannt. Auf die Indexregister kann man nur Operationen zum Laden, Speichern und Zählen anwenden.

Ein weiteres 8-Bit Register ist der Stackpointer (SP) (Stackpointer= Stapelzeiger). Er dient als Zeiger auf eine der Speicherstellen im Bereich $0100 bis $01FF. Dieser Bereich hat somit eine besondere Bedeutung. Hier können Rücksprungadressen und andere Werte abgelegt werden. Dazu gibt es zwei Operationen:

- Pusch speichert den Wert und vermindert dann den Stackpointer;
- Pull erhöht erst den Pointer und ließt dann den Wert.

Sie können sich den Stack als einen Stapel Zeitungen vorstellen. Mit Pusch legen Sie eine Zeitung oben auf den Stapel und mit Pull nehmen Sie die obere Zeitung wieder herunter.

Der zuletzt auf den Stapel gelegte Wert wird als erstes wieder gelesen!



Als letztes Register betrachten wir das Prozessorstatusregister (PSR). Es ist ebenfalls 8-Bit breit, jedoch haben hier die einzelnen Bits eine bestimmte Bedeutung. Die Bits werden daher als Flags, Flaggen oder Merker bezeichnet.

Bit 0: Carry Flag (C)

Dieses Bit setzt die CPU auf ‚1', wenn bei einer arithmetischen Operation ein Übertrag entsteht. Bei der Subtraktion wird das Bit umgekehrt genutzt: Es wird auf ‚0' gesetzt, falls ein Unterlauf auftritt.

Bit 1: Zero Flag (Z)

Dieses Bit setzt die CPU auf ‚1', wenn eine Operation Null ergibt.

Bit 2: Interrupt Flag (I)

Wird dieses Bit gesetzt, so werden Unterbrechungsanforderungen auf der IRQ Leitung des Prozessors ignoriert.

Bit 3: Dezimal Flag (D)

Ist dieses Bit gesetzt, arbeitet die CPU im Dezimal Modus.

Bit 4: Break Flag (B)

Wird von der CPU nach einem Break-Befehl gesetzt.

Bit 5: Unbenutzt

Bit 6: Overflow Flag (V)

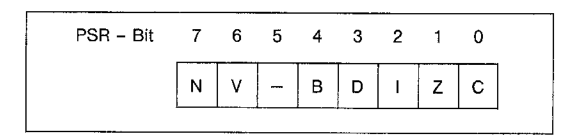
Wird von der CPU gesetzt, wenn bei einer arithmetischen Operation mit vorzeichenbehafteten Zahlen ein Fehler entsteht.

Bit 7: Negativ Flag

Wird von der CPU gesetzt, wenn sich bei einer Operation ein negatives Ergebnis ergibt.

(Abweichungen von den Regeln werden später bei den einzelnen Befehlen betrachtet.)

Das Prozessorstatusregister ist also folgendermaßen organisiert:



PSR – Bit	7	6	5	4	3	2	1	0
	N	V	–	B	D	I	Z	C

4/5.2.1.3

Die Hardware Vektoren RESET, IRQ und NMI

Nun wissen wir zwar wie die CPU aufgebaut ist, aber woher weiß die CPU nun, was sie zu tun hat? Betrachten wir also einmal, was nach dem Einschalten des Computers passiert. Beim Einschalten wird von einer Elektronik ein Signal auf eine spezielle Leitung, die RESET Leitung (Reset=Rücksetzen) gelegt. Dieses Signal erkennt die CPU. Sie lädt nun den Programmzähler PC mit dem Inhalt der Speicherzellen $FFFC und $FFFD. In diesen Speicherstellen muß also die Startadresse eines Programms stehen. Im C64 ist es die Adresse der Reset-Routine, deren Aufgabe es ist, den Bildschirm zu löschen, den Speicher zu testen und schließlich den BASIC-Interpreter (oder ein Modul) zu starten.

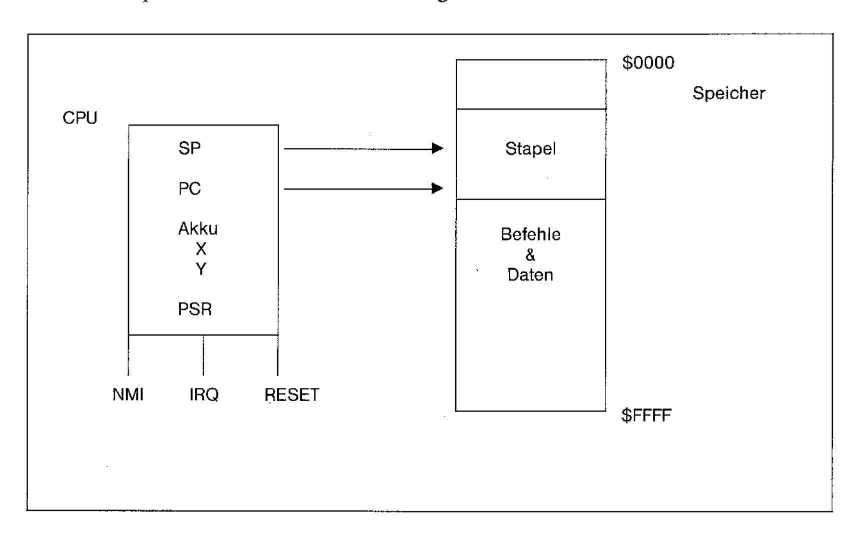
Es gibt aber noch zwei weitere Leitungen, die die CPU zu bestimmten Aktionen veranlassen. Es sind dies die Leitungen NMI und IRQ. Beide Leitungen sind Unterbrechungsanforderungen. Liegt an einer Leitung eine Unterbrechungsanforderung an, so unterbricht die CPU das laufende Programm, bringt die Adresse und das Prozessorstatusregister auf den Stapel (Stack) und springt in das Programm, dessen Startadresse sie in den Speicherstellen $FFFA/$FFFB bei NMI bzw. $FFFE/

$FFFF bei IRQ findet. Nach Beendigung des Programms, holt sich die CPU und PC und das Prozessorstatusregister wieder vom Stapel, und fährt mit der Programmausführung genau an der Stelle fort, an der sie unterbrochen wurde.

Im C64 wird die IRQ-Leitung dazu benutzt, alle 60stel Sekunde die Uhrzeit zu erhöhen und die Tastatur abzufragen. Die NMI-Leitung wird nur für die RS 232 C-Schnittstelle benutzt.

Der Unterschied zwischen der IRQ und der NMI-Leitung besteht darin, daß man der CPU verbieten kann, auf die IRQ-Leitung zu reagieren. Dies wird durch das Interrupt Flag im Prozessorstatusregister erreicht. Wird dieses Bit gesetzt, ignoriert die CPU die IRQ-Leitung. Die Reaktion auf die NMI-Leitung kann nicht verboten werden.

Unser Computermodell sieht also wie folgt aus:

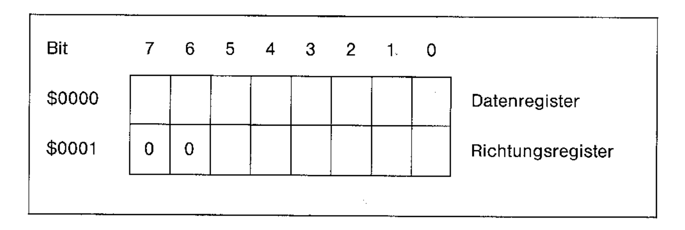


4/5.2.1.4

Der Prozessorport der 6510

Alles bisher gesagte gilt sowohl für die CPU 6502 als auch für die 6510. Wo liegt nun aber der Unterschied? Von der technischen Seite gesehen, hat die 6510 eine Reihe von Unterschieden. Diese interessieren uns als Programmierer nicht besonders. Für uns gibt es eigentlich nur einen Unterschied: Die Verwendung der Speicherstellen $0000/$0001. Bei der 6502 sind dies ganz normale Speicherstellen. Bei der 6510 ist hier jedoch ein 6-Bit Port realisiert. Das heißt, die CPU hat 6 frei programmierbare Leitungen, die im C64 jedoch eine bestimmte Funktion haben (siehe auch 2/2.1.6 PLA). Über die Speicherstelle $0000 werden die Leitungen eingelesen bzw. Signale auf die Leitungen gelegt und über die Speicherstelle $0001 festgelegt, ob es sich um eine Eingabeleitung (Bit=0) oder Ausgabeleitung (Bit=1) handelt. Dabei müssen die höherwertigen zwei Bit auf 0 (= Eingang) gelegt werden, da die 6510 intern die beiden oberen Bit für die RDY und die NMI-Leitung benutzt.

Prozessorport:



Bit	7	6	5	4	3	2	1	0	
$0000									Datenregister
$0001	0	0							Richtungsregister

Alle Leitungen finden im Commodore 64 Verwendung. Eine Modifikation der Daten kann daher zu einem Systemabsturz führen. Der Computer kann dann nur noch durch kurzes Ausschalten wieder zum Arbeiten gebracht werden. Der Prozessorport ermöglicht aber auch auf die 64K-Byte Speicher unseres Computers zuzugreifen, da über die unteren drei Bit die Speicheraufteilung gesteuert wird.

4/5.2.2

Die Programmierung der CPU

Nachdem die CPU als Baustein ausgiebig betrachtet wurde, werden wir uns jetzt der Programmierung der CPU zuwenden. Wir werden uns daher zunächst die Programm- und Befehlsstruktur ansehen.

4/5.2.2.1

Einführung

Einiges können wir aus dem bisher gesagten bereits über die Programmierung der CPU ableiten:

1. Das Programm muß als Folge von Bitmustern im Speicher liegen.
2. Befehle werden sich auf die Register Akku, X, Y etc. beziehen (abgesehen von Sprungbefehlen).

Wir wollen uns im Folgenden eine kleine Auswahl von Befehlen ansehen, die unsere CPU ausführen kann.

Ein häufiger Befehl heißt: Lade den Akku. Er hat das Bitmuster 10101001 oder hexadezimal $A9. Seine Funktion ist recht einfacht: der Inhalt der nach dem Befehl folgenden Speicherstelle wird in den Akku übertragen. Also lädt die Folge $A9 $4E den Akku mit dem Wert 78 (=$4E). Nun ist es aber äußerst lästig immer ‚lade den Akku‘ oder gar $A9 zu sagen, wenn man den Akku laden will. ‚Lade den Akku‘ ist zu lang, und ‚$A9‘ sagt nichts über den Befehl aus. Man führt deshalb Abkürzungen ein, die man sich leicht merken kann (sogenannte mnemotechnische Bezeichnungen).

Will man ein Register laden, so schreibt man LD für laden und A für Akku. Somit haben wir die Befehle LDA (lade Akku), LDX (lade X-Register) und LDY (lade Y-Register). Da man Register im allgemeinen nicht nur laden, sondern auch speichern will, benötigt man Store- (Speicher) Befehle: STA, STX und STY.

In Bitmusterdarstellung entspricht STA 10001101 ($ 8D). Nun muß noch die Adresse folgen. Da eine Adresse 16 Bit hat, muß diese in den folgenden zwei Byte stehen. Die 6510 verlangt, daß bei einer Adresse zuerst der niederwertige Teil, und dann der höherwertige folgt.

Nehmen wir zum Beispiel einmal an, daß Sie den Buchstaben ‚B' in die linke obere Ecke des Bildschirms bringen wollen. In BASIC können Sie das mit POKE 1024,2 erreichen. In Maschinensprache funktioniert das genauso. Wir laden zuerst eines der Register Akku, X oder Y mit 2 und speichern den Wert dann ab.

```
Also: LDA 2
      STA 1024
```

oder in Bitmusterschreibweise $A9 $02 $8D $00 $04 (1024=$0400).

An diesem Beispiel kann man auch erkennen, daß unser bisheriges System, Maschinenprogramme in für uns lesbarer Form darzustellen, noch unzureichend ist. Man kann (und will) mit LDA nämlich nicht nur Werte, die direkt hinter dem Befehl stehen in den Akku laden, sondern auch die Werte aus einer beliebigen Speicherzelle. Wie unterscheidet man dies? Üblich ist, das LDA 2 den Wert in der Speicherstelle mit der Adresse 2 meint, wie ja auch bei STA 1024 die Speicherstelle gemeint ist. Um anzuzeigen, daß der Wert direkt dem Befehl folgt, stellen wir ihm ein ‚#' voran: LDA #2. In der Bitmusterschreibweise ist die Sache schwieriger. Hier wird LDA #2 wie gehabt mit $A9 $02, aber LDA 2 mit $A5 $02 übersetzt. Diese Schreibweise wollen wir im Folgenden aber nicht weiter betrachten. Wenn Sie diese Codes interessieren, so schlagen Sie sie bitte in der Tabelle nach.

4/5.2.2.2

Beispiel Addition

Als nächstes wollen wir in einem Beispiel zwei Zahlen addieren. Die 6510 kennt nur einen Additionsbefehl: ADC (add with carry / addiere mit Übertrag). Er addiert zum Inhalt des Akkumulators den Operanden (d.h. den direkt folgenden Wert oder den Inhalt einer Speicherstelle) und das Carry-Bit des Prozessorstatusregisters. Um zwei Zahlen addieren zu können, muß also zunächst des Carry-Bit des PSR auf Null gesetzt werden. Dies tut der Befehl CLC (Clear Carry).

Der umgekehrte Befehl zum Setzen des Carry-Bit heißt übrigens SEC (SEt Carry). Ebenso gibt es analoge Befehle zum Setzen und Löschen der PSR-Bits D,I und V: CLD, CLI, CLV, SED und SEI, auf die wir in Kapital 4/5.2.4 noch zu sprechen kommen. Das folgende kleine Programm berechnet nun die Summe des Inhalts der Speicherstelle $C000 mit der Zahl 100 und speichert den Wert wieder in die Speicherstelle $C000:

```
CLC             Carry-Bit auf 0 setzen
LDA  $C000      Inhalt der Speicherzelle $C000 in den Akku bringen
ADC #100        Akku+100+Carry-Bit in den Akku bringen
STA  $C000      Akkuinhalt wieder in $C000 abspeichern
```

Da der Akkumulator aber nur 8-Bit breit ist und er nur Zahlen von 0 bis 255 oder von -128 bis +127 annehmen kann, müssen wir betrachten, was bei Überschreitung des Bereichs passiert.

Angenommen wir arbeiten nur mit ganzen Zahlen. Dann kann der Akku Zahlen von 0 bis 255 darstellen. Ist der Inhalt von $C000 also kleiner als 156 geht alles gut. Es entsteht kein Übertrag, also bleibt das Carry-Bit (welches bekanntlich einen Übertrag anzeigt) auf Null. Ist der Inhalt von $C000 größer als 155 z.B. 160, so erhalten wir als Ergebnis 160+100=260. Im Akku steht jedoch nur 4, da er nur 8 Bit zur Verfügung hat (260-256=4) und das Carry-Bit ist auf Eins gesetzt.

Eine 16-Bit Addition sieht also wie folgt aus:

```
CLC             Übertrag löschen
LDA  $C000      Zahl 1   niederwertiger Teil in Akku
ADC  $C002      + Zahl 2 niederwertiger Teil
STA  $C004      Als niederwertiger Teil Ergebnis speichern
LDA  $C001      Zahl 1   höherwertiger Teil in Akku. Hier wird das Carry
                nicht gelöscht, um den richtigen Übertrag zu erhalten
ADC  $C003      + Zahl 2 höherwertiger Teil + Übertrag
STA  $C005      Als höherwertiger Teil Ergebnis speichern
```

Wobei die Zahlen in $C000/$C001 bzw. $C002/$C003 und das Ergebnis in $C004/ $C005 gemäß der 6510-Reihenfolge abgespeichert sind.

Bei vorzeichenbehafteten Zahlen geht man genauso vor. Hier ist jedoch zu beachten, daß das höchstwertigste Bit (egal ob 8 oder 16-Bit Zahl) das Vorzeichen der Zahl darstellt. Ein Übertrag in diese Stelle bei der Addition hat deshalb eine Verfälschung des Vorzeichens zur Folge, sodaß das Ergebnis falsch ist. Tritt dieser Fall ein, setzt die CPU das V-Bit im PSR auf Eins, sonst auf Null. Um dies

noch einmal zu verdeutlichen ein Beispiel: Die Zahlen 64 und 70 sollen addiert werden. Dies geschieht wie folgt:

```
  01000000                  +  64
+ 01000110                 + + 70
__________ oder dezimal _______
= 10000110                 = – 122 FALSCH
```

Die Subtraktion funktioniert genauso, nur heißt hier der Befehl SBC statt ADC (subtract with carry) und das Carry-Bit wird genau umgekehrt genutzt (d.h. Carry = 1 bedeutet kein Übertrag). Ein 16-Bit Subtraktionsprogramm sieht dann so aus:

```
SEC          Übertrag löschen
LDA $C000    Zahl 1   niederwertiger Teil in Akku
SBC $C002    – Zahl 2 niederwertiger Teil
STA $C004    Als niederwertiger Teil Ergebnis speichern
LDA $C001    Zahl 1   höherwertiger Teil in Akku
SBC $C003    – Zahl 2 höherwertiger Teil – Übertrag
STA $C005    Als höherwertiger Teil Ergebnis speichern
```

Wobei die Zahlen wieder in $C000/$C001 bzw. $C002/$C003 und das Ergebnis in $C004/$C005 gemäß der 6510-Reihenfolge abgespeichert sind.

4/5.2.2.3

Beispiel Zeichensatz

Wir wollen uns noch ein zweites Beispiel ansehen, bei dem wir den Zeichensatz des C 64 auf dem Bildschirm anzeigen. Dazu werden wir folgendermaßen vorgehen: Der C 64 hat einen Zeichensatz von 256 Zeichen, die von 0 bis 255 numeriert sind. Dies sind genau die Nummern, die ein Register darstellen kann. Das X-Register dient uns daher als Zähler. Diese Nummer übertragen wir dann in den Akku und speichern es auf den Bildschirm. Dann wird der Inhalt des X-Registers erhöht, bis wir wieder auf Null landen.

Zuerst brauchen wir aber ein paar neue Befehle:

Der Befehl, der den Inhalt des X-Registers in den Akku bringt heißt TXA (transfer X into accumulator). Weiter stellt uns der 6510 die Transportbefehle TAX, TAY, TSX, TXS und TYA zur Verfügung, wobei S den Stackpointer bezeichnet.

Dann gibt es noch den Befehl INX (increment X) der das X-Register um 1 erhöht (INY erhöht entsprechend das Y-Register). Es gibt auch das Gegenteil: DEX (decrement X) bzw. DEY der das X bzw. Y-Register um 1 erniedrigt.

Damit können wir bereits einen Versuch wagen:

```
     LDX #0        X-Register mit Null laden
(*)  TXA           X-Register in den Akku übertragen
     STA 1024,X    Auf den Bildschirm bringen
     INX           X Register erhöhen
```

In der dritten Zeile steht STA 1024,X. Dies bedeutet, daß der Wert im Akku in die Speicherstelle geschrieben werden, die die Adresse 1024+Inhalt des X-Registers hat. Die CPU 6510 ist in der Lage diesen Befehl auszuführen. Also wird Zeichen 0 in 1024+0, Zeichen 1 in 1024+1 etc. gespeichert. Ein direktes STX 1024,X geht mit der CPU 6510 jedoch nicht! Ebenso gibt es keinen Befehl INA!

Etwas fehlt aber noch in unserem Programm. Nach dem Erhöhen des X-Registers müssen wir an der mit (*) bezeichneten Stelle weitermachen, falls das X-Register ungleich Null ist, oder anders ausgedrückt bei der Erhöhung des X-Registers das Ergebnis nicht Null war. Wir brauchen also den Befehl ‚springe wenn das Z-Bit im PSR gleich Null ist‘ oder kurz BNE (branch on not equal to zero). Als Argument verlangt der BNE-Befehl die Sprungweite (d.h. die Anzahl der zu überspringenden Byte) als vorzeichenbehaftete 8-Bit Zahl. Man kann mit ihm also maximal −126 bis +129 Byte springen (zwei Byte zählt der Befehl, daher von −128+2 bis +127+2). Daher müssen wir unser Programm doch wieder in Bitmusterdarstellung betrachten, um die Sprungweite ausrechnen zu können:

```
$a2 $00       LDX#0         X-Register mit Null laden
$8a           TXA           X-Register in den Akku übertragen
$9d $00 $04   STA 1024,X    Auf den Bildschirm bringen
$e8           INX           X Register erhöhen
$d0 $f9       BNE -7        Sprung zum TXA Befehl, falls X ungleich Null
```

Damit haben wir unser Ziel erreicht, aber wir mußten dabei wieder auf Bitmusterebene arbeiten. Dies wollen wir in Zukunft vermeiden. Wir werden deshalb das Sprungziel mit einem frei gewählten Namen bezeichnen und bei dem Sprungbefehl einfach den Namen angeben. In unserem Beispiel sieht das dann so aus:

```
          LDX #0         X-Register mit Null laden
SCHLEIFE: TXA            X-Register in den Akku übertragen
          STA 1024,X     Auf den Bildschirm bringen
          INX            X Register erhöhen
          BNE SCHLEIFE   Sprung zum TXA Befehl, falls X ungleich Null
```

Nachdem wir uns nun soviel mit Theorie beschäftigt haben, wollen wir auch das Programm einmal von Basic aus ausführen. Da das Programm unabhängig von der Startadresse ist, wählen wir hier willkürlich $C000. Wir müssen nun die Bitmuster von Basic aus in den Speicherbereich ab $C000 bringen. Dazu werden die Bitmuster in einer DATA-Zeile dezimal abgelegt, da Basic nur Dezimalzahlen kennt. Dann werden in einer FOR...NEXT-Schleife die Werte gelesen und mit POKE in den Speicherbereich gebracht. Dann muß das Programm noch gestartet werden. Dies wird mit dem Befehl SYS49152 erreicht ($C000=49152). Das entsprechende BASIC-Programm sieht dann wie folgt aus:

```
1 PRINT CHR$(147);: FOR A=0 TO 300: PRINT ".";: NEXT
2 FOR A=49152 TO 49161: READ W: POKE A,W: NEXT
3 SYS 49152
4 DATA 162,0,138,157,0,4,232,208,249,96
5 REM =$A2,$00,$8A,$9D,$00,$04,$E8,$D0,$F9,$60
```

Das Programm beinhaltet noch zum Schluß den Befehl RTS (= $60), der den Rücksprung zu Basic veranlaßt, aber für das Verständnis des Programms im Moment unwichtig ist. Die Zeile 1 dient nur zum Füllen des Farbspeichers, denn ohne Farbe sieht man auf dem C 64 bekanntlich nichts.

Starten Sie nun das Programm mit RUN.

4/5.2.2.4

BCD-Arithmetik

Die CPU 6510 kann noch etwas ganz besonderes: BCD Arithmetik (BCD= binary coded decimal). Um in BCD-Darstellung arbeiten zu können, müssen wir das Dezimal-Bit D im PSR mit dem Befehl SED setzen. Die Befehle ADC und SBC arbeiten nun in BCD-Arithmetik. Das bedeutet, daß in jedem Byte eine zweistellige Dezimalzahl von 00 bis 99 dargestellt werden kann (packed BCD). Also nehmen je vier Bit (= ein Nibble) eine der Zahlen 0 bis 9 auf.

Beispiel: die Zahl 35 wird binär als 00100011, aber in BCD als 00110101 dargestellt.
! 3!! 5!

Die Befehle ADC und SBC arbeiten nun bei gesetzten D-Bit völlig problemlos mit dieser Zahlendarstellung. Sie werden wie wir es gewohnt sind verwendet. Also:

```
8-Bit Addition   8-Bit Subtraktion
CLC              SEC
LDA Zahl 1       LDA Zahl 1
ADC Zahl 2       SBC Zahl 2
STA Ergebnis     STA Ergebnis
```

Zur Verdeutlichung noch ein Zahlenbeispiel:

```
  16                    00010110
+ 45                  + 01000101
____   oder binär:    __________
= 61                  = 01100001
```

4/5.2.3

Die Adressierungsarten der 6510

Mit Adressierungsart bezeichnet man die Art, durch die der Operand eines Befehls ermittelt wird. Drei Adressierungsarten haben wir bereits in unseren Beispielen kennengelernt.

4/5.2.3.1

Implizierte Adressierung

Bei der implizierten Adressierung ist die Adressierungsart bereits im Befehl enthalten (impliziert). Beim 6510 sind dies ausschließlich Befehle, die keinen Operanden benötigen, also z.B. DEY, CLC usw. Diese Befehle belegen alle genau ein Byte Speicherplatz.

4/5.2.3.2

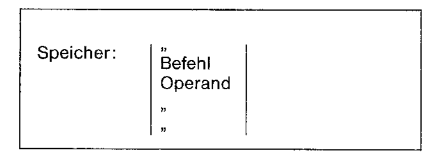
Adressierung des Akkumulators

Bei der Akkumulator-Adressierung ist der Inhalt des Akkumulators der Wert, auf den der Befehl angewendet werden soll. Bei der 6510 gibt es nur die Befehle ASL, LSR, ROL und ROR die in dieser Adressierungsart möglich sind. Die Befehle arbeiten jedoch auch in anderen Adressierungsarten. In Assemblern wird diese Adressierung häufig mit ASL A oder einfach mit ASL ohne Operand gekennzeichnet.

4/5.2.3.3

Unmittelbare Adressierung

Bei der unmittelbaren Adressierung folgt der Datenwert direkt dem Befehl. Diese Adressierungsart wird mit einem ‚#' gekennzeichnet (z.B. LDA #33, um den Akkumulator mit 33 zu laden).



4/5.2.3.4

Absolute Adressierung

Hier folgt dem Befehl nicht der Datenwert, sondern die Adresse, in der der Datenwert steht. Da eine Adresse 16-Bit breit ist, muß sie in den zwei dem Befehl folgenden Byte abgelegt sein. Der Prozessor verlangt, daß der niederwertige Teil zuerst abgelegt wird. In Assemblerschreibweise wird diese Adressierungsart nicht gekennzeichnet (z.B. LDA $2A12). Hier wird also an der fehlenden Kennzeichnung die Adressierungsart erkannt.

Speicher	"	
	Befehl	
	Adresse	Adresse niederwertiger Teil
	Adresse	Adresse höherwertiger Teil
	"	
	"	

4/5.2.3.5

Indiziert-absolute Adressierung

Bei dieser Adressierung setzt sich die Adresse des Datenwertes aus zwei Werten zusammen. Zu der hinter dem Befehl angegebenen Adresse wird der Inhalt des X oder des Y Registers vorzeichenlos addiert. Dies ergibt die Adresse unter der der Datenwert abgelegt ist. Die Assemblerschreibweise ist LDA adresse, X bzw. LDA adresse, Y. Diese Adressierungsart wird verwendet um Tabellen zu verwalten. Dabei wird als Adresse die Startadresse der Tabelle angegeben und das X- (bzw. Y-) Register als Zeiger auf einen Tabellenwert benutzt. Es können so Tabellen mit bis zu 256 Werten verwaltet werden.

4/5.2.3.6

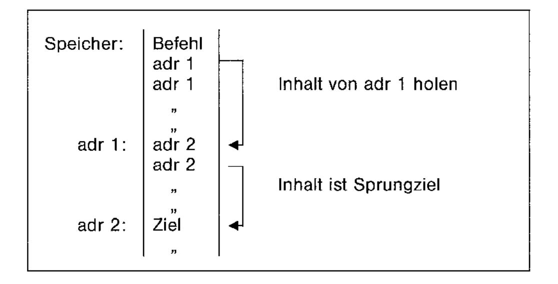
Indirekte Adressierung

Bei der indirekten Adressierung wird als Operand eine Adresse adr1 angegeben. Unter dieser Adresse steht in zwei Byte gemäß der 6510-Reihenfolge wieder eine Adresse adr2. Diese Adresse ist die Adresse für die Ausführung des Befehls. In Assemblerschreibweise wird die Adresse adr1 in Klammern angegeben. Also z.B. JMP ($8003). Die indirekte Adressierung ist nur bei dem Befehl JMP erlaubt.

Beispiel: Die CPU tritt auf den Befehl JMP ($C000) und die Adressen $C000 und $C001 seien wie folgt belegt:

$C000: $34
$C001: $12

Dann passiert folgendes: Die CPU holt die Adresse, die unter $C000 abgelegt ist, also $1234 und springt zu dieser Adresse (JMP $1234).



4/5.2.3.7

Zero-Page Adressierung

Die Zero-Page-Adressierung ist eine Kurzform der absoluten Adressierung. Die ersten 256 Adressen (d.h. die Adressen $0000 bis $00FF) können mit dieser Kurzform adressiert werden. Dabei wird statt der gesamten Adresse nur der niederwertige Teil angegeben, da der höherwertige Teil immer $00 ist. Diese Adressierungsart belegt daher nur zwei statt drei Byte zusammen mit dem Byte, für den Befehl gerechnet, und ist wesentlich schneller. Der Assembler erkennt die Form im allgemeinen selbständig. Eine einheitliche Kennzeichnung bei den Assembler, wo dies nicht der Fall ist, gibt es leider nicht.

4/5.2.3.8

Zero-Page indiziert

Diese Adressierungsart entspricht der Indiziert-absoluten Adressierung (4/5.2.3.5). Jedoch werden für die Adressen $0000–$00FF wieder nur zwei Byte benötigt (Befehlsbyte und ein Byte Adresse).

4/5.2.3.9

Zero-Page indiziert/indirekt

Bei dieser Adressierungsart werden die indirekte und die indizierte Adressierung miteinander verknüpft. Dabei bietet die 6510 zwei Möglichkeiten:

Vorindizierung: Assemblersyntax: LDA (adr,X)

Zu der angegebenen Zero-Page-Adresse adr wird der Inhalt des X-Registers vorzeichenlos addiert: Dies ergibt eine Adresse adr2, die in der Zero-Page stehen muß. Unter dieser Adresse steht die Adresse adr3 des Datenwertes. Die Adresse muß als 16-Bit Adresse gemäß der 6510 Reihenfolge in der Zero-Page stehen. Diese Adressierungsart funktioniert nur mit dem X-Register.

Nachindizierung: Assemblersyntax: LDA (adr,Y)

Unter der angegebenen Zero-Page-Adresse adr muß eine 16-Bit Adresse adr2 gemäß der 6510 Reihenfolge stehen. Zu dieser Adresse wird der Inhalt des Y-Registers vorzeichenlos addiert. Das Resultat ist die Adresse adr3 des Datenwertes. Diese Adressierungsart funktioniert nur mit dem Y-Register.

4/5.2.3.10

Relative Adressierung

Hier wird als Adresse eine 8-Bit Distanz angegeben. Die Adresse errechnet sich dann nach der Formel:

Adresse = Programmzähler + Distanz

wobei die Distanz eine vorzeichenbehaftete Zahl ist. Diese Adressierungsart wird nur bei bedingten Sprungbefehlen benutzt (z.B. BNE, BEQ, BVC etc.). Daher ergibt sich die maximale Sprungweite von –126 bis +129 (siehe auch Beispiel Zeichensatz). Einem Assembler wird einfach die Adresse mitgeteilt, zu der man springen will. Er errechnet die Distanz automatisch.

4/5.2.4

Mnemonics — Die Befehle der 6510

Nachdem in den Beispielen bereits viele Befehle der CPU 6510 mit ihrem mnemotechnischen Bezeichnungen vorgestellt wurden, folgt nun eine Gesamtliste der Befehle. Die Befehle sind dabei in logisch zusammengehörige Gruppen eingeteilt.

Zur weiteren Verdeutlichung sind zu den Befehlen kleine Basic-Unterprogramme angegeben, welche die Befehle simulieren. Diese Programme werden später zu einem Simulationsprogramm für 6510 Maschinenprogramme zusammengebunden (vgl). Kapitel 4/5.2.6). In den Programmen wird davon ausgegangen, daß die Adresse des Operanden in der Variablen AD, und der Wert, der unter dieser Adresse steht, in der Variablen WE bereits von einer Programmablaufsteuerung abgelegt wurden.

Alle Mnemonics werden in Teil 11 nochmals in einer Tabelle zusammengefaßt.

Die Basicprogramme verwenden folgende Variablen:

Variable	Inhalt	Bedeutung
AC	0..255	Akkumulator
XR	0..255	X-Register
YR	0..255	Y-Register
SR	0..255	Statusregister
SP	0..255	Stackpointer
SP%(255)	0..255	Array zur Simulation des Stackbereichs
N	0..1	Negativ Flag
V	0..1	Overflow Flag
B	0..1	Break Flag
D	0..1	Dezimal Flag
I	0..1	Interrupt Flag
Z	0..1	Zero Flag
C	0..1	Carry Flag
WE	0..255	Wert des Operanden
AD	0..65535	Adresse des Operanden
OP	0..255	Operationscode des Befehls (Opcode)
PC	0..65535	Programmzähler
AM%(255)	1..12	Enthält die Nummer der Adressierungsart des Operationscodes

Die Adressierungsarten wurden wie folgt durchnumeriert:

Nr.	Adressierungsart
1	Akkumulator oder Implizit
2	Unmittelbar
3	Relativ
4	Indirekt
5	(Indirekt,X)
6	(Indirekt),Y
7	Absolut
8	Zero-Page
9	Absolut,X
10	Zero-Page,X
11	Absolut,Y
12	Zero-Page,Y

4/5.2.4.1

Lade- und Speicherbefehle

Die Befehle dieser Gruppe dienen dem Laden und Speichern der Register der CPU. Sie sind in den Beispielen bereits behandelt worden.

LDA — LoaD Accu with memory

Dieser Befehl gehört zu den am häufigsten benötigten Befehlen. Der Akkumulator wird mit dem Inhalt der angegebenen Speicherzelle geladen. Die Flaggen Z (zero) und N (negativ) werden entsprechend dem geladenen Wert gesetzt. Alle anderen Flaggen bleiben unverändert.

Folgende Adressierungsarten sind erlaubt:

Unmittelbar	LDA #Wert
Absolut	LDA Adresse
Absolut,X	LDA Adresse,X
Absolut,Y	LDA Adresse,Y
Zero-Page	LDA Zero
Zero-Page,X	LDA Zero,X
(indirekt,X)	LDA (Zero,X)
(indirekt),Y	LDA (Zero),Y

Basic-Programm:

```
1220 REM ** LDA **
1230 AC=WE
1240 N=—(AC>127):Z=—(AC=0):RETURN
```

LDX — LoaD X-Register with memory

Das X-Register wird mit dem Inhalt der angegebenen Speicherzellen geladen. Die Flaggen Z (zero) und N (negativ) werden wieder entsprechend dem geladenen Wert gesetzt. Der Befehl entspricht also dem LDA-Befehl, nur, daß statt des Akkumulators das X-Register als Zielregister verwendet wird.

Erlaubte Adressierungsarten:

Unmittelbar	LDX #Wert
Absolut	LDX Adresse
Absolut,Y	LDX Adresse,Y
Zero-Page	LDX Zero
Zero-Page,Y	LDX Zero,Y

Basic-Programm:

```
1250 REM ** LDX **
1260 XR=WE
1270 N=—(XR>127):Z=—(XR=O):RETURN
```

LDY — LoaD Y-Register with memory

Entspricht dem Befehl LDA, jedoch wird das Y-Register mit dem Inhalt der angegebenen Speicherzelle geladen. Die Flaggen Z (zero) und N (negativ) werden entsprechend dem geladenen Wert gesetzt.

Erlaubte Adressierungsarten:

Unmittelbar	LDY #Wert
Absolut	LDY Adresse
Absolut,X	LDY Adresse,X
Zero-Page	LDY Zero
Zero-Page,X	LDY Zero,X

Basic-Programm:

```
1280 REM ** LDY **
1290 YR=WE
1300 N=—(YR>127):Z=—(YR=0):RETURN
```

STA — STore Accu in memory

Der Inhalt des Akkumulators wird unter der angegebenen Adresse abgelegt. Dies ist die Umkehrung des LDA-Befehls, mit dem der Akkumulator geladen wird.

Es werden keine Flaggen manipuliert.

Erlaubte Adressierungsarten:

Absolut	STA Adresse
Absolut,X	STA Adresse,X
Absolut,Y	STA Adresse,Y
Zero-Page	STA Zero
Zero-Page,X	STA Zero,X
(indirekt),X)	STA (Zero,X)
(indirekt),Y	STA (Zero),Y

Basic-Programm:

```
1920 REM ** STA **
1930 POKE AD,AC:RETURN
```

STX — STore X-Register in memory

Der Inhalt des X-Registers wird unter der angegebenen Adresse abgelegt. Der Befehl entspricht dem STA-Befehl, jedoch wird das X-Register verwendet. Außerdem sind die erlaubten Adressierungsarten gegenüber dem STA-Befehl stark eingeschränkt. Auch hier werden keine Flaggen manipuliert.

Erlaubte Adressierungsarten:

Absolut	STX Adresse
Zero-Page	STX Zero
Zero-Page,Y	STX Zero,Y

Basic-Programm:

```
1940 REM ** STX **
1950 POKE AD,XR:RETURN
```

STY — STore Y-Register in memory

Entspricht dem STX-Befehl, jedoch wird hier der Inhalt des Y-Registers unter der angegebenen Adresse abgelegt. Flaggen werden ebenfalls nicht manipuliert.

Erlaubte Adressierungsarten:

Absolut	STY Adresse
Zero-Page	STY Zero
Zero-Page,X	STY Zero,X

Basic-Programm:

```
1960 REM ** STY **
1970 POKE AD,YR:RETURN
```

4/5.2.4.2

Zählbefehle

Die Befehle dieses Abschnitts dienen zum Erhöhen oder Erniedrigen von Registern oder Speicherstellen um eine Einheit. Sie dienen also zum Zählen und zur Schleifenbildung. Im Beispiel Zeichensatz (4/5.2.2.3) finden Sie eine Anwendung.

DEC — DECrement memory

Der Inhalt der angegebenen Speicherstelle wird um eine Einheit vermindert. Wird dabei der Wert Null erreicht, wird das Zero-Flag gesetzt. Bei negativen Ergebnissen wird das N-Flag gesetzt. Alle anderen Flaggen bleiben unverändert.

Erlaubte Adressierungsarten:

Absolut	DEC Adresse
Absolut,X	DEC Adresse,X
Zero-Page	DEC Zero
Zero-Page,X	DEC Zero,X

Basic-Programm:

```
950 REM ** DEC **
960 WE=WE—1:IF WE<0 THEN WE=255:POKE AD,WE
970 PONE AB, WE:N=—(WE>127):Z=—(WE=0):RETURN
```

DEX — DEcrement X-Register

Hier wird der Inhalt des X-Registers um eine Einheit vermindert. Da der DEC-Befehl nur im Speicher arbeitet, sind für die Register die Spezialbefehle DEX und DEY nötig. Die Flaggen werden wie bei dem DEC-Befehl gesetzt.

Erlaubte Adressierungsarten:

nur implizit

Basic-Programm:

```
980 REM ** DEX **
990 XR=XR—1:IF XR<0 THEN XR=255
1000 N=—(XR>127):Z=—(XR=0):RETURN
```

DEY — DEcrement Y-Register

Entspricht dem DEX-Befehl. Hier wird jedoch der Inhalt des Y-Registers um eins vermindert. Für den Akkumulator existiert kein Decrement-Befehl. Die Flaggen werden ebenfalls wie beim DEC-Befehl gesetzt.

Erlaubte Adressierungsarten:

nur implizit

Basic-Programm:

```
1010 REM ** DEY **
1020 YR=YR—1:IFYR<0THENYR=255
1030 N=—(YR>127):Z=:(YR=0):RETURN
```

INC — INCrement memory

Der Inhalt der angegebenen Speicherstelle wird um eine Einheit erhöht. Wird dabei der Wert Null erreicht, wird das Zero-Flag gesetzt. Bei negativen Ergebnissen wird das N-Flag gesetzt. Der INC-Befehl ist die Umkehrung des DEC-Befehls.

Erlaubte Adressierungsarten:

Absolut	INC Adresse
Absolut,X	INC Adresse,X
Zero-Page	INC Zero
Zero-Page,X	INC Zero,X

Basic-Programm:

```
1070 REM ** INC **
1080 WE=(WE+1) AND 255:POKEAD,WE
1090 N=—(WE>127):Z=-(WE=0):RETURN
```

INX — INcrement X-Register

Analog zu den Decrement-Befehlen existieren bei den Increment-Befehlen auch zwei Befehle für das X- und Y-Register. Bei dem INX-Befehl wird der Inhalt des X-Registers um eine Einheit erhöht. Wird dabei der Wert Null erreicht, wird das Zero-Flag gesetzt. Bei negativen Ergebnissen wird das N-Flag gesetzt.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
1100 REM ** INX **
1110 XR=(XR+1) AND 255
1120 N=-(XR>127):Z=-(XR=0):RETURN
```

INY — INcrement Y-Register

Entspricht dem INX-Befehl. Hier wird jedoch der Inhalt des Y-Registers um eine Einheit erhöht. Die Flaggen werden wie beim INC-Befehl gesetzt. Ein Befehl zum Erhöhen des Akkumulators existiert, analog zu den Decrement-Befehlen nicht.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
1130 REM ** INY **
1140 YR=(YR+1) AND 255
1150 N=-(YR>127):Z=-(YR=0):RETURN
```

Das folgende Beispiel tauscht die Daten der Bildschirmseite des C 64 mit dem Speicherbereich ab $C100. Es schaltet also bei jedem Aufruf zwischen zwei Bildschirmseiten um.

```
temp:      .eq 2          ;Zwischenspeicher
zeiger1:   .eq 251        ;Zeiger auf Originalbildschirm
zeiger2:   .eq 253        ;Zeiger auf Speicher
bild1:     .eq $400       ;Originalbildschirm
bild2:     .eq $c100      ;Speicher

           .ba $c000      ;Basisadresse
           .os            ;Ausgabe in das RAM
```

```
start:     lda #<bild1         ;Zeiger1 auf Org. Bild
           ldx #>bild1
           sta zeiger1
           stx zeiger1+1
           lda #<bild2         ;Zeiger2 auf Speicher
           ldx #>bild2
           sta zeiger2
           stx zeiger2+1
           ldx #4              ;4 Seiten a 256 Byte
           ldy #0              ;Seitenindex
transfer:  lda (zeiger1),y     ;Originalwert..
           sta temp            ;merken
           lda (zeiger2),y     ;Speicherwert..
           sta (zeiger1),y     ;in Bildschirm
           lda temp            ;Originalwert..
           sta (zeiger2),y     ;in Speicher
           iny                 ;nächstes Byte
           bne transfer
           inc zeiger 1+1      ;nächste Seite
           inc zeiger 2+1
           dex
           rts
```

Die verwendeten Pseudo-Befehle '.BA','.OS' und '.EQ' sind vom verwendeten Assembler abhängig. Die hier angegebenen Befehle gelten für den in diesem Buch verwendeten Assembler und sind im Kapitel 4/6.4.2.6 beschrieben.

4/5.2.4.3

Logische Befehle

Dieses Kapitel beinhaltet die logischen Befehle der CPU 6510. Es gibt neben den Grundfunktionen UND, ODER und EXCLUSIVE-ODER Befehle zum Rechts- und Linksschieben, sowie einen Bit-Testbefehl. Logische Befehle arbeiten ausnahmslos mit dem Akkumulator zusammen.

Für die Und (AND), oder (OR) und Exclusive-Order (XOR) Funktion ist hier noch einmal die Wahrheitstabelle angegeben:

A	B	A AND B	A OR B	A XOR B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

AND — AND accu with memory

Der Inhalt des Akkumulators wird bitweise mit dem Inhalt der angegebenen Speicherstelle Und-verknüpft. Das Ergebnis steht im Akkumulator. Ist das Ergebnis der Operation Null, wird das Zero-Flag gesetzt. Bei einem negativen Ergebnis wird das Negativ-Flag gesetzt. Alle anderen Flaggen bleiben unverändert.

Beispiel:

Akku:	10101101
Speicher:	00001111
Ergebnis:	00001101

Erlaubte Adressierungsarten:

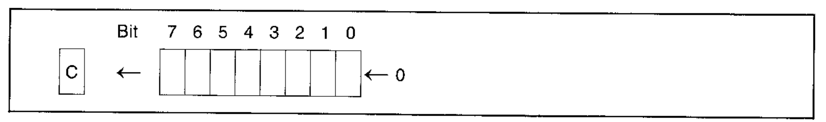
Unmittelbar	AND #Wert
Absolut	AND Adresse
Absolut,X	AND Adresse,X
Absolut,Y	AND Adresse,Y
Zero-Page	AND Zero
Zero-Page,X	AND Zero,X
(indirekt,X)	AND (Zero,X)
(indirekt),Y	AND (Zero),Y

Basic-Programm:

```
270 REM ** AND **
280 AC=(AC AND WE)
290 N=—(AC>127):Z=—(AC=0)
300 RETURN
```

ASL — Arithmetic Shift Left

Der Inhalt des Akkumulators oder der angegebenen Speicherstelle wird bitweise um eine Stelle nach links geschoben. Das höchstwertigste Bit wird in das Carry-Bit übertragen. In das niederwertigste Bit wird eine Null eingeschoben. Dies entspricht einer vorzeichenlosen Multiplikation mit zwei. Ist das Resultat eine Null, so wird das Zero-Flag gesetzt. Bei einem negativen Ergebnis wird das Negativ-Flag gesetzt. Die anderen Flaggen bleiben unverändert.



Mit Hilfe dieses Befehles kann man auch die häufig benötigte Multiplikation mit 10 einfach erreichen:

```
MULT 10:ASL       ;Akku * 2
        STA HELP  ;Ergebnis merken
        ASL       ;ergibt Akku * 4
        ASL       ;ergibt Akku * 8
        CLC
        ADC HELP  ;Akku * 8 + Akku * 2
                  ; = Akku * 10
```

Erlaubte Adressierungsarten:

Akkumulator	ASL
Absolut	ASL Adresse
Absolut,X	ASL Adresse,X
Zero-Page	ASL Zero
Zero-Page,X	ASL Zero,X

Basic-Programme:

```
310  REM ** ASL **
320  IFAM%(OP)<>1THEN340
330  AC=AC*2:C=—(AC>255):AC=AC AND 255
     :Z=—(AC=0):N=—(AC>127):RETURN:REM ASL AKKU
340  WE=WE*2:C=—(WE>255):WE=WE AND 255
     :Z=—(WE=0):N=—(WE>127):REM ASL ADRESSE
350  POKEAD,WE:RETURN
```

BIT — BIT Test

Dieser Befehl dient zum Testen der Bits der angegebenen Speicherstelle. Er wird immer dann eingesetzt, wenn im Programm Flaggen im Speicher getestet werden sollen.

Der Bit-Befehl löst folgende Aktionen aus:

— Der Akkumulator und der Inhalt der Speicherstelle werden Und-verknüpft. Das Zero-Flag Z wird gesetzt, falls das Ergebnis Null ist.

— Bit 6 der Speicherstelle wird in das V-Flag übertragen.

— Bit 7 der Speicherstelle wird in das N-Flag übertragen

Erlaubte Adressierungsarten:

Absolut	BIT Adresse
Zero-Page	BIT Zero

Basic-Programm:

```
480 REM ** BIT **
490 N=-(WE>127):V=(WE AND 64)/64
500 Z=-((AC AND WE)=0)
510 RETURN
```

Da der Bit-Befehl nur Flaggen, aber keine Register beeinflußt, wird er auch häufig für den folgenden Programmiertrick benutzt:

```
EINS:   LDA #1          ;Akku :=1
        .BY $2C         ;Code für BIT absolut
ZWEI:   LDA #2          ;Akku :=2
        STA SPEICHER    ;Akku speichern
```

Der hier verwendete Pseudo-Befehl '.BY $2C' legt den Hexadezimalwert $2C in einem Byte im Speicher ab (vgl. Kapitel 4/5.4.2.6). Er ist abhängig von dem verwendeten Assembler und kann daher in fremden Programmlistings anders aussehen.

Je nachdem, ob die Routine bei EINS oder ZWEI angesprungen wird, wird 1 oder 2 in der Speicherstelle SPEICHER abgelegt.

Das funktioniert so:

— Wird die Routine bei ZWEI angesprungen, ist alles klar.

— Wird die Routine bei EINS angesprungen, wird zunächst der Akku mit 1 geladen. Dann stößt die CPU auf den Code $2C und führt den Bit-Befehl aus. Als Adresse werden die folgenden zwei Byte benutzt, in denen der Befehl 'LDA #2' steht. Nach dem Bit-Befehl wird also beim Befehl 'STA SPEICHER' weitergearbeitet. Da der Akku immer noch 1 enthält, wird eine 1 in SPEICHER abgelegt.

EOR — Exclusive-OR accu with memory

Der Inhalt des Akkumulators wird bitweise mit dem Inhalt der angegebenen Speicherstelle Exclusive-Oder-verknüpft. Das Ergebnis steht im Akkumulator. Mit diesem Befehl können genau bestimmte Bits im Akkumulator umgedreht werden. Bei einer Null als Ergebnis wird das Zero-Flag gesetzt. Bei einem negativen Ergebnis wird das Negativ-Flag gesetzt. Die anderen Flaggen werden nicht verändert.

Beispiel:

```
Akku:       10101101
Speicher:   00001111
Ergebnis:   10100010 (die letzten vier Stellen sind umgedreht)
```

Da die CPU 6510 leider keinen NOT-Befehl zum Umdrehen aller Bits kennt, muß dieser durch den Befehl 'EOR #$FF' ersetzt werden.
Mit dem EOR-Befehl können auch zwei Bitmuster vertauscht werden. Stehen die Bitmuster unter der Adresse A bzw. B, so vertauscht die folgende Befehlsfolge die beiden Bitmuster:

```
swap:   LDA A     ;A:= A EOR B
        EOR B
        STA A
        EOR B     ;B:= A EOR B
        STA B
        EOR A     ;A:= A EOR B
        STA A
```

Um die Arbeitsweise deutlich zu machen ist hier noch ein Zahlenbeispiel angegeben:

Befehl	A	B	Akku
Startwerte	10101010	11001100	00000000
LDA A	10101010	11001100	10101010
EOR B	10101010	11001100	01100110
STA A	01100110	11001100	01100110
EOR B	01100110	11001100	10101010
STA B	01100110	10101010	10101010
EOR A	01100110	10101010	11001100
STA A	11001100	10101010	11001100

Erlaubte Adressierungsarten:

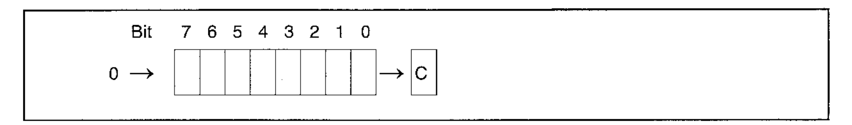
Unmittelbar	EOR #Wert
Absolut	EOR Adresse
Absolut,X	EOR Adresse,X
Absolut,Y	EOR Adresse,Y
Zero-Page	EOR Zero
Zero-Page,X	EOR Zero,X
(indirekt,X)	EOR (Zero,X)
(indirekt),Y	EOR (Zero),Y

Basic-Programm:

```
1040 REM ** EOR **
1050 AC=((AC AND NOT WE) OR (WE AND NOT AC))
1060 N=-(AC>127):Z=-(AC=0):RETURN
```

LSR — Logical Shift Right

Der Inhalt des Akkumulators oder der angegebenen Speicherstelle wird bitweise um eine Stelle nach rechts geschoben. Das niederwertigste Bit wird in das Carry-Bit übertragen. In das höchstwertigste Bit wird eine Null eingeschoben. Dies entspricht einer vorzeichenlosen Division durch zwei, und ist die Umkehrung des ASL-Befehls. Ist das Ergebnis Null, wird das Zero-Flag gesetzt. Ein negatives Ergebnis kann bei diesem Befehl nicht erreicht werden, da Bit 7 immer auf Null gesetzt wird. Daher wird auch das Negativ-Flag immer gelöscht. Andere Flaggen werden nicht verändert.



Erlaubte Adressierungsarten:

Akkumulator	LSR
Absolut	LSR Adresse
Absolut,X	LSR Adreesse,X
Zero-Page	LSR Zero
Zero-Page,X	LSR Zero,X

Basic-Programm:

```
1310 REM ** LSR **
1320 IF AM%(OP)<>1 THEN 1350
1330 C=AC AND 1:AC=INT (AC/2):REM LSR AKKU
1340 N=-(AC>127):Z=-(AC=0):RETURN
1350 C=WE AND 1:WE=INT (WE/2):POKEAD,WE:REM LSR ADRESSE
1360 N=-(WE>127):Z=-(WE=0):RETURN
```

ORA — OR Accu with memory

Der Inhalt des Akkumulators wird bitweise mit dem Inhalt der angegebenen Speicherstelle ODER-verknüpft. Das Ergebnis steht im Akkumulator. Mit diesem Befehl können genau bestimmte Bits im Akkumulator gesetzt werden. Die Flaggen werden wie bei dem AND-Befehl gesetzt.

Beispiel:

Akku:	10101101
Speicher:	00001111
Ergebnis:	10101111

Erlaubte Adressierungsarten:

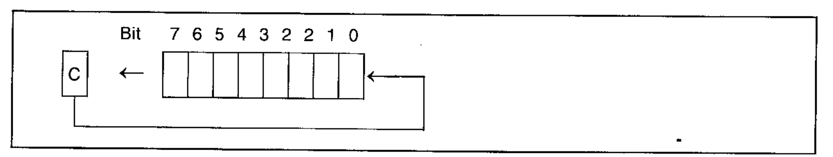
Unmittelbar	ORA #Wert
Absolut	ORA Adresse
Absolut,X	ORA Adresse,X
Absolut,Y	ORA Adresse,Y
Zero-Page	ORA Zero
Zero-Page,X	ORA Zero,X
(indirekt,X)	ORA (Zero,X)
(indirekt),Y	ORA (Zero),Y

Basic-Programm:

```
1390  REM ** ORA **
1400  AC=AC OR WE
1410  N=-(AC>127:Z=-(AC=0):RETURN
```

ROL — ROtate Left one bit

Der Inhalt des Akkumulators oder der angegebenen Speicherstelle wird bitweise um eine Stelle nach links geschoben. In das niederwertigste Bit wird das Carry-Bit übertragen. Das höchstwertigste Bit wird danach in das Carry-Bit übertragen. Ist die Carry-Flagge gelöscht, entspricht der Befehl dem ASL-Befehl. Ist das Ergebnis der Operation Null, wird das Zero-Flag gesetzt. Bei einem negativen Ergebnis wird das Negativ-Flag gesetzt. Die anderen Flaggen bleiben unverändert.



Mit Hilfe dieses Befehles kann man auch mehr als 8-Bit-Werte linksschieben:

```
MULT:    ASL WERT      ;erste 8-Bit schieben
         ROL WERT+1    ;Bit 8-15 schieben
         ROL WERT+2    ;Bit 16-23 schieben

WERT:    .BY 0,0,0     ;24-Bit-Wert
```

Der Befehl '.BY' in dem Beispiel ist ein Pseudo-Befehl (vgl. Kapitel 4/5.4.2.6). Er hat hier die Aufgabe drei Byte mit Null zu füllen. In diesen drei Byte wird der 24-Bit-Wert angenommen.

Erlaubte Adressierungsarten:

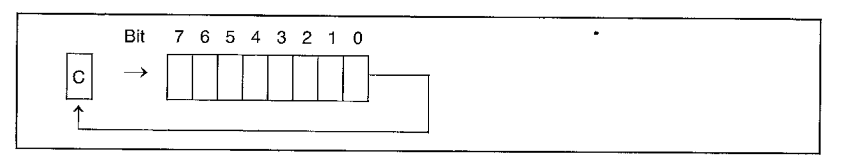
Akkumulator	ROL
Absolut	ROL Adresse
Absolut,X	ROL Adresse,X
Zero-Page	ROL Zero
Zero-Page,X	ROL Zero,X

Basic-Programm:

```
1500  REM ** ROL **
1510  IF AM%(OP)<>1 THEN 1540
1520  AC=AC*2+C:C=—(AC>255):AC=ACAND255
      :REM ROL AKKU
1530  N=—(AC>127):Z=—(AC=0):RETURN
1540  WE=WE*2+C:C=—(WE>255):WE=WEAND 255
      :REM ROL ADRESSE
1550  N=—(WE>127):Z=—(WE=0):POKEAD,WE:RETURN
```

ROR — ROtate Right one bit

Der Inhalt des Akkumulators oder der angegebenen Speicherstelle wird bitweise um eine Stelle nach rechts geschoben. In das höchstwertigste Bit wird das Carry-Bit übertragen. Das niederwertigste Bit wird danach in das Carry-Bit übertragen. Bei gelöschter Carry-Flagge entspricht dies dem LSR-Befehl. Die Flaggen werden wie beim ROL-Befehl gesetzt.



Analog zum ROL-Befehl kann man mit Hilfe dieses Befehls auch mehr als 8-Bit-Werte rechtsschieben:

```
DIVU:   LSR WERT+2    ;Bit 16-23 rechtsschieben
        ROR WERT+1    ;Bit  8-15 rechtsschieben
        ROR WERT      ;Bit  0-7  rechtsschieben

WERT:   .BY 0,0,0     ;24-Bit-Wert
```

Hier wird wieder der Pseudo-Befehl '.BY' eingesetzt, um Platz für den 24-Bit-Wert zu schaffen (vgl. ROL-Befehl).

Erlaubte Adressierungsarten:

Akkumulator	ROR
Absolut	ROR Adresse
Absolut,X	ROR Adresse,X
Zero-Page	ROR Zero
Zero-Page,X	ROR Zero,X

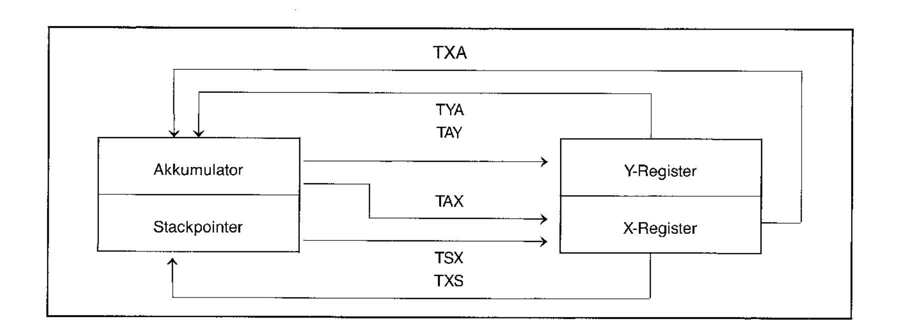
Basic-Programm:

```
1560  REM ** ROR **
1570  IF AM%(OP)<>1 THEN 1600
1580  H=AC AND 1:AC=INT (AC/2)+C*128
      :C=H:REM ROR AKKU
1590  N=—(AC>127):Z=—(AC=0):RETURN
1600  H=WE AND 1:WE=INT (WE/2)+C*128
      :C=H:REM ROR ADRESSE
1610  N=—(WE>127):Z=—(WE=0):POKEAD,WE:RETURN
```

4/5.2.4.4

Transferbefehle

Die Transferbefehle gehören mit zu den einfachsten Befehlen der CPU 6510. Sie dienen zum Transport von Daten zwischen zwei Registern der CPU. Eine Übersicht über die Möglichkeiten der Transferbefehle soll die folgende Grafik bieten:



TAX — Transfer Accu into X-Register

Der Inhalt des Akkumulators wird in das X-Register übertragen. Die Flaggen Z (zero) und N (negativ) werden entsprechend dem übertragenen Wert gesetzt.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
1980 REM ** TAX **
1990 XR=AC:Z=—(XR=0):N=—(XR>127):RETURN
```

TAY — Transfer Accu into Y-Register

Der Inhalt des Akkumulators wird in das Y-Register übertragen. Die Flaggen Z (zero) und N (negativ) werden entsprechend dem übertragenen Wert gesetzt. Der Befehl entspricht also dem TAX-Befehl, nur daß das Y-Register Ziel der Operation ist.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
2000 REM ** TAY **
2010 YR=AC:Z=—(YR=0):N=—(YR>127):RETURN
```

TSX — Transfer Stackpointer into X-Register

Der Inhalt des Stackpointers (SP) wird in das X-Register übertragen. Die Flaggen Z (zero) und N (negativ) werden entsprechend dem übertragenen Wert gesetzt. Dies ist die einzige Möglichkeit, den Wert des Stackpointers zu lesen.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
2020 REM ** TSX **
2030 XR=SP:Z=-(XR=0):N=-(XR>127):RETURN
```

TXA — Transfer X-register into Accu

Der Inhalt des X-Registers wird in den Akkumulator übertragen. Die Flaggen Z (zero) und N (negativ) werden entsprechend dem übertragenen Wert gesetzt. Dies ist die Umkehrung des TAX-Befehls. Die Befehle TAX und TXA können z.B. zur Zwischenspeicherung des Akkumulators in dem X-Register verwendet werden. Häufig soll aber auch auf dem X-Register (oder Y-Register) ein logischer oder arithmetischer Befehl angewendet werden. Da diese Befehle nur mit dem Akkumulator zusammenarbeiten, muß der Inhalt erst in den Akkumulator mit TXA transportiert, dann im Akkumulator verarbeitet, und schließlich mit TAX wieder in das X-Register zurückgebracht werden.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
2040 REM ** TXA **
2050 AC=XR:Z=-(AC=0):N=-(AC>127):RETURN
```

TXS — Transfer X-register into Stackpointer

Der Inhalt des X-Registers wird in den Stackpointer übertragen. Die Flaggen Z (zero) und N (negativ) werden entsprechend dem übertragenen Wert gesetzt. Dies ist die einzige Möglichkeit, dem Stackpointer einen Wert zuzuweisen. Jedes Lesen oder Schreiben des Stackpointers geht also immer nur über das X-Register.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
2060 REM ** TXS **
2070 SP=XR:Z=-(SP=0):N=-(SP>127):RETURN
```

TYA — Transfer Y-register into Accu

Der Inhalt des Y-Registers wird in den Akkumulator übertragen. Die Flaggen Z (zero) und N (negativ) werden entsprechend dem übertragenen Wert gesetzt. Dies ist die Umkehrung des TAY-Befehls.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
2080  REM ** TYA **
2090  AC=YR:Z=-(AC=0):N=-(AC>127):RETURN
```

4/5.2.4.5

Arithmetische Befehle

Als arithmetische Befehle kennt die CPU 6510 nur die Addition und Subtraktion. Diese können aber sowohl im gepackten BCD-Modus, als auch im Binär-Modus arbeiten.

ADC — ADd with Carry

Der Inhalt der angegebenen Speicherstelle und das Carry-Bit wird zum Akkumulator addiert. Bei einem Übertrag wird das Carry-Bit gesetzt. Das Overflow-Flag wird gesetzt, wenn der Bereich für eine vorzeichenbehaftete Zahl überschritten wird. Ist das D-Flag gesetzt, arbeitet der Befehl in BCD-Arithmetik. Ergibt sich als Ergebnis Null, wird das Zero-Flag gesetzt. Bei einem negativen Ergebnis wird das Negativ-Flag gesetzt. Beispiele für die Addition finden Sie in den Kapiteln 4/5.2.2.2 und 4/5.2.2.4.

Erlaubte Adressierungsarten:

Unmittelbar	ADC #Wert
Absolut	ADC Adresse
Absolut,X	ADC Adresse,X
Absolut,Y	ADC Adresse,Y
Zero-Page	ADC Zero
Zero-Page,X	ADC Zero,X
(indirekt,X)	ADC (Zero,X)
(indirekt),Y	ADC (Zero),Y

Basic-Programm:

```
140  REM ** ADC **
150  IF D THEN 220:REM DEZIMALMODUS AN
160  V=(AC AND 127)+(WE AND 127)+C:REM V FLAGGE BERECHNEN
170  V=—(V>127)
180  AC=AC+WE+C:C=—(AC>255):AC=AC AND 255:N=—(AC>127):Z=—(AC=0)
190  V=(C AND NOT V) OR (V AND NOT C)
200  RETURN
210  REM ADC IN DEZIMALMODUS
220  H=(WE AND 15)+(AC AND 15)+C:REM UNTERES NIBBLE
230  C=—(H>9):H=H—10*C
240  AC=(WE AND 240)+(AC AND 240)+C*16+H:REM OBERES NIBBLE
250  C=—(AC>153):AC=AC—160*C:N=—(AC>127):Z=—(AC=0)
260  RETURN
```

Als weiteres Beispiel sei hier noch ein Programm angegeben, das eine 16-Bit Binärzahl in eine 24-Bit BCD-Zahl umwandelt:

```
TOBCD:     LDA #0          ;BCD-Zahl auf Null setzen
           STA BCD
           STA BCD+1
           STA BCD+2
           LDX #16         ;Zähler für 16 Bit
BCDLOOP:   ASL ZAHL        ;16-Bit Zahl mal 2
           ROL ZAHL+1      ;Bit 16 in Carry
           SED             ;Dezimal Mode an
           LDA BCD         ;BCD-Zahl*2+Bit 16 (Carry)
           ADC BCD
           STA BCD
           LDA BCD+1
           ADC BCD+1
           STA BCD+1
           LDA BCD+2
           ADC BCD+2
           STA BCD+2
           CLD             Dezimal Mode aus
           DEX             ;Bitzähler -1
           BNE BCDLOOP     ;bis 16 Bit
           RTS             ;fertig
ZAHL:      .BY 0,0         ;Hier wird die 16-Bit Zahl abgelegt
BCD:       .BY 0,0,0       ;und hier die 24-Bit BCD-Zahl
```

Hier wird wieder der Pseudo-Befehl '.BY'benutzt, um Platz für die verwendeten Zahlen zu schaffen.

SBC — SuBtract with Carry

Vom Akkumulator wird der Inhalt der angegebenen Speicherstelle subtrahiert. War das Carry-Bit nicht gesetzt, wird außerdem eins subtrahiert. Bei einem Übertrag

wird das Carry-Bit gelöscht. Das Overflow-Flag wird gesetzt, wenn der Bereich für eine vorzeichenbehaftete Zahl überschritten wird. Ist das Ergebnis der Subtraktion Null, wird das Zero-Flag gesetzt. Bei einem negativen Ergebnis wird das Negativ-Flag gesetzt. Die anderen Flaggen werden nicht verändert. Ist das D-Flag gesetzt, arbeitet der Befehl in BCD-Arithmetik.

Erlaubte Adressierungsarten:

Unmittelbar	SBC #Wert
Absolut	SBC Adresse
Absolut,X	SBC Adresse,X
Absolut,Y	SBC Adresse,Y
Zero-Page	SBC Zero
Zero-Page,X	SBC Zero,X
(indirekt,X)	SBC (Zero,X)
(indirekt),Y	SBC (Zero),Y

Basic-Programm:

```
1730 REM ** SBC **
1740 IF D THEN 1810
1750 H=WE+1—C:V=0:REM SBC BINAER
1760 IFAC<128THENIFH>127ANDH<129+ACTHENV=1:REM V FLAGGE BERECHNEN
1770 IFAC>127THENIFH>AC—128ANDH<128THENV=1
1780 AC=AC—WE—1+C:C=—(AC>=0):Z=—(AC=0):AC=AC AND 255:N=—(AC>127)
1790 RETURN
1800 REM SBC IM BCD—MODUS
1810 H=(AC AND 15)—(WE AND 15)—1+C
1820 C=—(H>=0):H=(H AND 15)—6+6*C
1830 AC=(AC AND 240)—(WE AND 240)—16+16*C
1840 C=—(AC>=0):AC=(AC AND 240)—96+96*C+H
1850 Z=—(AC=0):N=—(AC>127):RETURN
```

Befehle zur Multiplikation oder Division kennt die CPU 6510 wie gesagt leider nicht. Sie müssen per Programm nachgebildet werden. Im folgenden finden Sie ein Programm, das zwei 16-Bit-Zahlen multipliziert. Der Algorithmus folgt der normalen, schriftlichen Multiplikation, und ist daher leicht zu verstehen.

```
mult:       lda #0          ;ergebnis = 0
            sta ergebnis
            sta ergebnis+1
            ldx #16         ;Zähler für 16 Bit
multloop:   lsr zahl2+1     ;zahl 2 rechtsschieben (/2)
            ror zahl2       ;Bit 0 in's Carry
            bcc noadd       ;Bit war 0-> keine Addition
            clc             ;ergebnis + zahl1
            lda ergebnis
            adc zahl1
            sta ergebnis
            lda ergebnis+1
            adc zahl 1+1
            sta ergebnis+1
```

```
noadd:      asl zahl1         ;zahl1 linksschieben (*2)
            rol zahl1+1
            dex               ;Bitzähler -1
            bne multloop      ;Bis 16 Bit durch
            rts               ;fertig
zahl1:      .by 0,0           ;16-Bit Zahl 1
zahl2:      .by 0,0           ;16-Bit-Zahl 2
ergebnis:   .by 0,0           ;16-Bit Ergebnis
```

Auch in diesem Beispiel wird der Pseudo-Befehl '.BY' verwendet, um Platz für die verwendeten Zahlen zu schaffen.

4/5.2.4.6

Stackbefehle

Die Befehle dieses Kapitels dienen der Verwaltung des Stacks. Mit Hilfe dieser Befehle werden Werte auf den Stack gelegt, oder von ihm heruntergenommen. Die CPU 6510 erlaubt nur den Inhalt des Akkumulators oder des Prozessorstatusregisters auf den Stack abzulegen. Inhalte anderer Register oder Speicherstellen können daher nur über den Akkumulator auf den Stack gebracht werden.

PHA — PusH Akku on stack
Der Inhalt des Akkumulators wird auf den Stack gelegt. Dann wird der Stackpointer SP um eins vermindert. Mit diesem Befehl können Zwischenergebnisse für eine spätere Weiterverwendung auf den Stack abgelegt werden. Es werden keine Flaggen manipuliert.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
1420  REM ** PHA **
1430  SP%(SP)=AC:SP=SP—1:RETURN
```

PHP — PusH Processor status

Der Inhalt des Statusregisters wird auf den Stack gelegt. Dann wird der Stackpointer SP um eins vermindert.

Es werden keine Flaggen manipuliert.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
1440  REM ** PHP **
1450  SP%CSP)=SR:SP=SP—1:RETURN
```

PLA — PuLl Akku from stack

Der Stackpointer SP wird um eins erhöht. Dann wird ein Wert vom Stack geholt, und dem Akkumulator zugewiesen. Ein mit PHA auf den Stack gelegter Wert wird so in den Akkumulator zurückgeholt. War der Wert Null, so wird das Zero-Flag gesetzt. Bei einem negativen Wert wird das Negativ-Flag gesetzt.

Erlaubte Adressierungarten:
nur implizit

Basic-Programm:

```
1460  REM ** PLA **
1470  SP=SP+1:AC=SP%(SP):RETURN
```

PLP — PuLl Processor status

Der Stackpointer SP wird um eins erhöht. Dann wird ein Wert vom Stack geholt, und dem Prozessorstatusregister zugewiesen. Ein Flaggenzustand kann also mit PHP auf den Stack gerettet, und später mit PLP wieder zurückgeholt werden. Mit Hilfe der Stackbefehle kann auch dem Statusregister ein bestimmter Wert zugewiesen werden:

```
LDA #Wert     ;Wert in den Akkumulator laden
PHA           ;Wert auf den Stack bringen
PLP           ;Statusregister mit Wert laden
```

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
1480  REM ** PLP **
1490  SP=SP+1:SR=SP%(SP):RETURN
```

4/5.2.4.7

Flaggenmanipulation

Diese Befehle haben Sie bereits in den Beispielen kennengelernt. Sie dienen zum Setzen oder Löschen von Flaggen im Prozessorstatusregister.

CLC — CLear Carry flag

Das Carry-Flag im Prozessorstatusregister wird gelöscht. Dieser Befehl ist im allgemeinen vor einem ADC-Befehl nötig, da die Addition bei der 6510 immer mit Übertrag erfolgt:

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
780  REM ** CLC **
790  C=0:RETURN
```

CLD — CLear Dezimal mode

Das Dezimal-Flag im Prozessorstatusregister wird gelöscht. Die Befehle ADC und SBC arbeiten dann mit der Zweierkomplementdarstellung.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
800  REM ** CLD **
810  D=0:RETURN
```

CLI — CLear Interrupt mask

Das Interrupt-Flag im Prozessorstatusregister wird gelöscht. Unterbrechungsanforderungen auf dem IRQ-Eingang werden damit erlaubt. Dieser Befehl wird gebraucht, um nach einem SEI-Befehl die Interruptanforderungen wieder zuzulassen.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
820  REM ** CLI **
830  I=0:RETURN
```

CLV — CLear oVerflow flag

Das Overflow-Flag V im Prozessorstatusregister wird gelöscht. Dieser Befehl ist eigentlich überflüssig, da das V-Flag nur von den Befehlen ADC und SBC gesetzt und gelöscht werden braucht. Da die CPU 6510 jedoch ein Nachfolger der CPU 6502 ist, wurde der Befehl übernommen. Bei der CPU 6502 gibt es einen zusätzlichen Eingang, über den extern das V-Flag gesetzt werden kann (z.B. für Steuerungsaufgaben). Daher war hier der CLV-Befehl sehr nützlich.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
840  REM ** CLV **
850  V=0:RETURN
```

SEC — SEt Carrry flag

Das Carry-Flag im Prozessorstatusregister wird gesetzt. Dieser Befehl wird im allgemeinen vor einem SBC-Befehl verwendet, um den Übertrag bei der Subtraktion zu löschen.

Erlaubte Adressierungsarten:
nur implizit
Basic-Programm:

```
1860  REM ** SEC **
1870  C=1:RETURN
```

SED — SEt Dezimal mode

Das Dezimal-Flag im Prozessorstatusregister wird gesetzt. Die CPU arbeitet dann im gepackten BCD-Modus, wobei in einem Byte jeweils zwei Ziffern untergebracht sind.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
1880  REM ** SED **
1890  D=1:RETURN
```

SEI — SEt Interrupt mask

Das Interrupt-Flag im Prozessorstatusregister wird gesetzt. Unterbrechungsanforderungen auf der IRQ-Leitung werden damit verboten. Die NMI-Leitung kann nicht verboten werden. Um Interruptanforderungen wieder zuzulassen, muß der CLI-Befehl gegeben werden.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
1900  REM ** SEI **
1910  I=1:RETURN
```

4/5.2.4.8

Sprungbefehle

Die CPU 6510 kennt eine Reihe von bedingten Sprungbefehlen. Diese lassen jedoch nur Sprünge im Bereich von -126 bis +129 zu. Für längere Sprünge ist folgender, unbedingter Befehl zuständig:

JMP — JuMP to adress

Der Programmzähler wird mit dem angegebenen Wert geladen. Dies bewirkt einen Sprung. Dieser Befehl ist der einzige Befehl der CPU 6510, der die Addressierungsart indirekt beherrscht. Der Befehl beeinflußt keine Flaggen.

Erlaubte Adressierungsarten:

Absolut	JMP Addresse
(indirekt)	JMP (Adresse)

Basic-Programm:

```
1160  REM ** JMP **
1170  PC=AD:RETURN
```

BCC — Branch on Carry Clear

Das Carry-Flag wird getestet. Ist es gelöscht, wird der folgende Wert als vorzeichenbehaftete Zahl zu dem Programmzähler addiert, also ein Sprung ausgeführt. Auch hier werden keine Flaggen manipuliert.

Erlaubte Adresssierungsarten:
nur relativ

Basic-Programm:

```
360  REM ** BCC **
370  IF (SR AND 1) THEN RETURN
380  IF WE>127 THEN WE=WE—256
390  PC=PC+WE+2:RETURN
```

Eine Anwendung zeigt das folgende Beispiel, das den Inhalt des Akkumulators binär ausgibt:

```
          .ba $c000     ;Basisadresse
          .os           ;Ausgabe in's RAM
bsout:    .eq $ffd2     ;Ausgaberoutine des C 64
          ldx #8        ;8 Zeichen
bin:      rol           ;Akku links schieben, höchstes
                        ;Bit ins Carry-Bit
          pha           ;Akkuinhalt retten
          lda #'0       ;'0' laden
          bcc null      ;war höchsten Bit=0 dann '0'
          lda #'1       ;sonst '1' ausgeben
null:     jsr bsout
          pla           ;Akku wiederherstellen
          dex           ;Zeichenzähler-1
          bne bin       ;bis alle Zeichen ausgegeben
```

Die verwendeten Pseudo-Befehle '.BA' '.OS' und '.EQ' gelten für den in diesem Buch verwendeten Assembler und sind in Kapitel 4/5.4.2.6 beschrieben.

BCS — Branch on Carry Set

Das Carry-Flag wird getestet. Ist es gesetzt, wird der Sprung ausgeführt. Dieser Befehl wird z.B. zur Überprüfung des Ergebnisses einer vorzeichenlosen arithmetischen Operation benutzt. Es werden keine Flaggen manipuliert.

Erlaubte Adressierungsarten:
nur relativ

Basic-Programm:

```
400  REM ** BCS **
410  IF NOT(SR AND 1) THEN RETURN
420  IF WE>127 THEN WE=WE—256
430  PC=PC+WE+2:RETURN
```

BEQ — Branch on EQual to zero

Das Zero-Flag wird getestet. Ist es gesetzt, wird der Sprung ausgeführt. Der Befehl steht häufig hinter einem Vergleichsbefehl (z.B. CMP), um bei Gleichheit der Argumente eine bestimmte Routine abzuarbeiten, oder eine Schleife zu verlassen. Hier werden ebenfalls keine Flaggen manipuliert.

Erlaubte Adressierungsarten:
nur relativ

Basic-Programm:

```
440  REM ** BEQ **
450  IF NOT(SR AND 2) THEN RETURN
460  IF WE>127 THEN WE=WE—256
470  PC=PC+WE+2:RETURN
```

BMI — Branch on MInus

Das Negativ-Flag wird getestet. Ist es gesetzt, wird der Sprung ausgeführt. Eine Manipulation von Flaggen findet nicht statt.

Erlaubte Adressierungsarten:
nur relativ

Basic-Programm:

```
520  REM ** BMI **
530  IF NOT(SR AND 128) THEN RETURN
540  IFWE>127THENWE=WE—256
550  PC=PC+WE+2:RETURN
```

BNE — Branch on Not Equal to zero

Das Zero-Flag wird getestet. Ist es gelöscht, wird der Sprung ausgeführt. Es findet keine Flaggenmanipulation statt.

Erlaubte Adressierungsarten:
nur relativ

Basic-Programm:

```
560  REM ** BNE **
570  IF (SR AND 2) THEN RETURN
580  IF WE>127 THEN WE=WE—256
590  PC=PC+WE+2:RETURN
```

BPL — Branch on Plus

Das Negativ-Flag wird getestet. Ist es gelöscht, wird der Sprung ausgeführt. Eine Flaggenmanipulation wird nicht durchgeführt.

Erlaubte Adressierungsarten:
nur relativ

Basic-Programm:

```
600  REM ** BPL **
610  IF (SR AND 128) THEN RETURN
620  IF WE>127 THEN WE=WE—256
630  PC=PC+WE+2:RETURN
```

BVC — Branch on oVerflow Clear

Das Overflow-Flag wird getestet. Ist es gelöscht, wird der Sprung ausgeführt. Dieser Befehl wird im Zusammenhang mit dem SBC und ADC-Befehl bei vorzeichenbehafteter Arithmetik benötigt. Flaggen werden keine manipuliert.

Erlaubte Adressierungsarten:
nur relativ

Basic-Programm:

```
700  REM ** BVC **
710  IF (SR AND 64) THEN RETURN
720  IF WE>127 THEN WE=WE-256
730  PC=PC+WE+2:RETURN
```

BVS — Branch on oVerflow Set

Das Overflow-Flag wird getestet. Ist es gesetzt, wird der Sprung ausgeführt. Dieser Befehl wird im Zusammenhang mit dem SBC und ADC-Befehl bei vorzeichenbehafteter Arithmetik benötigt, um bei einer Bereichsüberschreitung eine Fehlerroutine anzuspringen. Eine Flaggenmanipulation findet nicht statt.

Erlaubte Adressierungsarten:
nur relativ

Basic-Programm:

```
740  REM ** BVS **
750  IF NOT (SR AND 64) THEN RETURN
760  IF WE>127 THEN WE=WE—256
770  PC=PC+WE+2:RETURN
```

4/5.2.4.9

Vergleichsbefehle

Zum Vergleichen von Zahlen stehen drei Befehle zur Verfügung, die jeweils eines der Register Accu, X- oder Y-Register mit dem Inhalt der angegebenen Speicherstelle vergleicht.

CMP — CoMPare to accumulator

Vom Akkumulator wird der Inhalt (M) der angegebenen Speicherstelle subtrahiert. Das Ergebnis wird jedoch nicht gespeichert, sondern nur die Flaggen N, Z und C gesetzt:

Z=1, falls die Werte übereinstimmen.

N=1, falls der Inhalt des Akkumulators kleiner als der Inhalt der Speicherstelle ist.

C=1, falls der Inhalt des Akkumulators größer oder gleich dem Inhalt der Speicherstelle ist.

Es ergibt sich somit folgende Tabelle:

Vergleich	N	Z	C
A<M	1 *	0	0
A=M	0	1	1
A>M	0 *	0	1

* Vergleich als vorzeichenbehaftete Zahlen

Erlaubte Adressierungsarten:

Unmittelbar	CMP #Wert
Absolut	CMP Adresse
Absolut,X	CMP Adresse,X
Absolut,Y	CMP Adresse,Y
Zero-Page	CMP Zero
Zero-Page,X	CMP Zero,X
(indirekt,X)	CMP (Zero,X)
(indirekt),Y	CMP (Zero),Y

Basic-Programm:

```
860 REM ** CMP **
870 C=—(AC>=WE):N=((AC—WE)AND128)/128
880 Z=—(AC=WE):RETURN
```

CPX — ComPare to X-Register

Vom X-Register wird der Inhalt (M) der angegebenen Speicherstelle subtrahiert. Das Ergebnis wird jedoch nicht gespeichert, sondern nur die Flaggen N, Z und C wie bei dem Befehl CMP gesetzt. Die Adressierungsarten sind gegenüber dem CMP-Befehl jedoch stark eingeschränkt.

Erlaubte Adressierungsarten:

Unmittelbar	CPX #Wert
Absolut	CPX Adresse
Zero-Page	CPX Zero

Basic-Programm:

```
890  REM ** CPX ***
900  C=—(XR>=WE):N=((XR-WE)AND128)/128
910  Z=—(XR=WE):RETURN
```

CPY — ComPare to Y-Register

Der CPY-Befehl entspricht dem CPX-Befehl. Jedoch wird hier das Y-Register für den Vergleich verwendet. Die Flaggen werden wie bei dem CMP-Befehl gesetzt.

Erlaubte Adressierungsarten:

Unmittelbar	CPY #Wert
Absolut	CPY Adresse
Zero-Page	CPY Zero

Basic-Programm:

```
920  REM ** CPY **
930  C=—(YR>=WE):N=((YR-WE)AND128)/128
940  Z=—(YR=WE):RETURN
```

Als Beispiel dient das folgende Programm. Es sortiert eine Folge von zehn Zahlen. Einmal werden die Zahlen als vorzeichenlos, das andere Mal als vorzeichenbehaftet angesehen. Die Ergebnisse der beiden Durchläufe finden Sie auf Seite 49.

```
            .ba $c000          ;Basisadresse
            .os                ;Ausgabe in's RAM
sort:       ldx #$ff           ;Index der aktuellen Zahl
loop1:      inx                ;Index+1
            txa                ; X—>Y
            tay                ;Y ist Index der zu vergl. Zahl
            cpx #10            ;Letzte Zahl?
            bne notready
            rts                ;ja, dann fertig
notready:   lda werte,x        ;aktuellen Wert holen
loop2:      iny                ;Prüfindex+1
            cpy #10            ;Letzte Zahl ?
            beq loop1          ;ja, dann nächste aktuelle Zahl
            cmp werte,y        ;Zahlen vergleichen
            bmi loop2          ;vorzeichenbehaftet kleiner
;           bcc loop2          für vorzeichenlos kleiner
            pha                ;Prüfwert < aktuelle Zahl
            lda werte,y        ;Werte vertauschen
            sta werte,x
            pla
            sta werte,x
            jmp notready       ;und weitertesten
werte:   .by -1,56,23,12,5,-5,-10,-77,34,-5 ;Zahlenfolge
```

Die verwendeten Pseudo-Befehle '.BY','.BA' und '.OS' sind im Kapitel 4/5.4.2.6 beschrieben.

Das Programm liefert folgende Ergebnisse:

Befehl	Reihenfolge										
bcc	05	0C	17	22	38	B3	F6	FB	FB	FF	Hexadezimal
bcc	5	12	23	34	56	179	246	251	251	255	Dezimal
bmi	B3	F6	FB	FB	FF	05	0C	17	22	38	Hexadezimal
bmi	—77	—10	—5	—5	—1	5	12	23	34	56	Dezimal

4/5.2.4.10

Strukturbefehle

Zur Abarbeitung von Unterprogrammen gibt es bei der CPU 6510 zwei Möglichkeiten. Als erstes können Unterprogramme über die Vektoren NMI und IRQ aufgerufen werden. Wichtiger für den Programmierer ist die Möglichkeit Unterprogramme per Befehl aufzurufen (z.B. Routinen des Betriebssystems).

JSR — Jump to SubRoutine

Der Programmzähler + 2 wird auf den Stack abgelegt. Dies ist nicht die Adresse des folgenden Befehls, sondern dessen Adresse — 1 ! Dann wird die angegebene Adresse in den Programmzähler geladen. Flaggen werden nicht manipuliert.

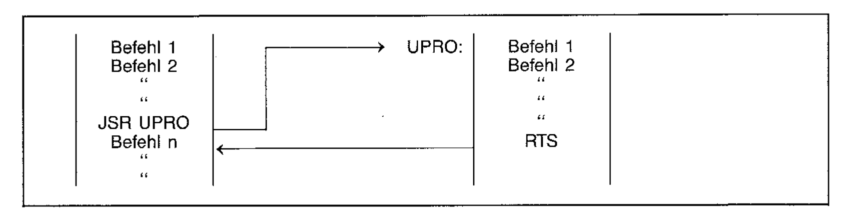
Erlaubte Adressierungsarten:
nur absolut

Basic-Programm:

```
1180  REM ** JSR **
1190  PC=PC+2:SP%(SP)=PC/256:SP=SP—1:REM PC+2 AUF STACK
1200  SP%(SP)=PC—256*INT(PC/256):SP=SP—1
1210  PC=AD:RETURN
```

RTS — ReTurn from Subroutine

Der Programmzähler wird vom Stack geladen, und um eins erhöhr. Flaggen werden dabei nicht manupuliert. Das Programm wird so bei dem Befehl, nach dem JSR-Befehl, fortgesetzt.



Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
1700  REM ** RTS **
1710  SP=SP+1:PC=SP%(SP):REM PC VOM STACK
1720  SP=SP+1:PC=PC+256*SP%(SP)+1:RETURN
```

Als Beispiel ist hier ein Unterprogramm angegeben, daß (selbst) wieder Unterprogramme aufruft. Es gibt eine 16-Bit Binärzahl hexadezimal aus:

```
;hexout gibt 16-Bit Zahl hexadezimal aus
bsout:      .eq $ffd2        ;Routine des C 64: gibt den Akku
                             ;als Zeichen aus
hexout:     lda #13          ;neue Zeile
            jsr bsout
            lda zahl+1       ;Höherwertiges Byte ausgeben
            jsr hexbyte
            lda zahl         ;Niederwertiges Byte ausgeben
            jsr hexbyte
            rts              ;fertig
;hexbyte gibt ein Byte hexadezimal aus
hexbyte:    pha              ;Byte merken
            lsr              ;Byte / 16
            lsr
            lsr
            lsr
            jsr nibble       ;obere 4 Bit als '0'..'f' ausgeben
            pla              ;Byte zurückholen
            and #%1111       ;untere 4 Bit als '0'..'f' ausgeben
            jsr nibble
            rts              ;fertig

;nibble gibt 4-Bit Wert als '0'..'f' aus
nibble:     clc              ;Wert in Zahl wandeln
            adc #'0
            cmp #'9+1        ;'9' überschritten ?
            bcc hout
            adc #6           ;ja, dann auf 'a'..'f' bringen
hout:       jsr bsout        ;und ausgeben
            rts              ;fertig
zahl:       .by 0,0          ;16-Bit Zahl
```

Die verwendeten Pseudo-Befehle '.EQ' und '.BY' sind in Kapitel 4/5.4.2.6 beschrieben.

RTI — ReTurn from Interrupt

Bei einer Unterbrechungsanforderung werden erst der Programmzähler, und dann das Prozessorstatusregister auf den Stack gerettet. Dann wird die Interruptroutine aufgerufen. Diese muß mit dem RTI-Befehl abgeschlossen werden. Dieser läd zunächst das Prozessorstatusregister wieder vom Stack, und dann den Programmzähler. Es werden also alle Flaggen auf den Stand vor der Unterbrechungsanforderung gebracht, und das Programm an der alten Stelle fortgesetzt.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
1620 REM ** RTI **
1630 SP=SP+1:SR=SP%(SP):REM STATUS VOM STACK
1640 SP=SP+1:PC=SP%(SP):REM PC VOM STACK
1650 SP=SP+1:PC=PC+256*SP%(SP)
1660 N=(SR AND 128) / 128:V=(SR AND 64) / 64
     :REM STATUS ZERLEGEN
1670 B=(SR AND 32) /32:D=(SR AND 8) /8
1680 I=(SR AND 4) /4:Z=(SR AND 2) /2:C=SR AND 1
1690 RETURN
```

Beim C 64 steht im RAM ein Vektor (ab $314), der auf die IRQ-Routine des Betriebssystems zeigt. Über diesen Vektor wird die IRQ-Routine aufgerufen, nachdem die Register der CPU gerettet worden sind. Sie können hier leicht eigene Routinen einbauen. Auch um den Abschluß der Routine brauchen Sie sich nicht zu kümmern. Entweder fahren Sie bei der alten IRQ-Routine fort (mit JMP $EA31), oder Sie beenden sie mit 'JMP $FEBC'. Das folgende Beispiel läßt alle Sterne ('*') auf dem Bildschirm blinken:

```
zeiger:     .eq 251          ;Zeiger auf Bildschirm
zähler:     .eq 253          ;Zähler für Blinkfrequenz
irqvec:     .eq $314         ;IRQ-Vektor
irqalt:     .eq $ea31        ;IRQ-Routine des C 64
bild:       .eq $400         ;Bildschirmadresse
            .ba $c000        ;Basisadresse
            .os              ;Ausgabe in das RAM
;Startroutine: baut die neue IRQ-Routine ein
init:       sei              ;Interrupt verbieten
            lda #<irqneu     ;IRQ-Vektor auf irqneu
            ldx #>irqneu
            sta irqvec
            stx irqvec+1
            cli              ;Interrupt wieder zulassen
            rts              ;fertig
;Neue Interrupt-Routine
irqneu:     inc zähler       ;Zähler+1
            lda #30          ;Zähler=30 ?
            cmp zähler
            bne fertig       ;nein, dann fertig
            lda #<bild       ;zeiger auf Bild setzen
            ldx #>bild
            sta zeiger
            stx zeiger+1
            ldx #4           ;4 Seiten a 256 Byte
            ldy #0           ;Byteindex
```

```
loop:       lda (zeiger),y     ;Zeichen holen
            and #$7f           ;Bit 7 auf Null
            cmp #"*            ;ist es ein Stern ?
            bne keinstern      ;nein
            lda (zeiger),y     ;ja, dann Originalwert holen..
            eor #$80           ;Bit 7 umdrehen..
            sta (zeiger),y     ;und wieder speichern
keinstern:  iny                ;Byteindex +1
            bne loop           ;weiter bis Index=0
            inc zeiger+1       ;neue Seite
            dex                ;Seitenzähler —1
            bne loop           ;bis 4 Seiten durch
            stx zähler         ;zähler auf Null
fertig:     jmp irqalt         ;weiter bei der Original-Routine
```

4/4.2.4.11

Sonstige Befehle

Zwei weitere Befehle kennt die CPU noch, die hauptsächlich bei der Fehlersuche und Programmentwicklung benutzt werden.

NOP — No OPeration

Dieser Befehl tut absolut nichts! Flaggen werden von dem Befehl auch nicht verändert. Er kann als Platzhalter im Programm eingesetzt werden, oder in Warteschleifen zur Verlängerung der Wartezeit eingesetzt werden.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
1370  REM ** NOP **
1380  RETURN
```

BRK — BReak

Mit dem BRK-Befehl kann per Programm ein Interrupt über den IRQ-Vektor ausgelöst werden. Es gibt jedoch zwei Unterschiede:

- Es wird der Programmzähler plus zwei auf den Stack gespeichert.
- Das B-Flag wird eingesetzt bevor das Statusregister auf den Stapel gelegt wird, um den BRK-Befehl von IRQ zu unterscheiden.

Außer dem Break-Flag werden keine Flaggen manipuliert.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
640  REM ** BRK **
650  PC = PC + 2:SP%(SP) = PC/256:SP = SP—1:REM PC + 2 AUF STACK
660  SP%(SP) = PC—256*INT(PC/256):SP = SP—1
670  B = 1:SR = SROR32:SP%(SP) = SR:SP = SP—1
     :REM STATUS AUF STACK
680  I = 1:PC = PEEK (65534) + PEEK (65535)*256
     :REM WEITER BEI INTERRUPT
690  RETURN
```

BRK — BReak

Mit dem BRK-Befehl kann per Programm ein Interrupt über den IRQ-Vektor ausgelöst werden. Es gibt jedoch zwei Unterschiede:

— Es wird der Programmzähler plus zwei auf den Stack gespeichert.

— Das B-Flag wird eingesetzt bevor das Statusregister auf den Stapel gelegt wird, um den BRK-Befehl von IRQ zu unterscheiden.

Außer dem Break-Flag werden keine Flaggen manipuliert.

Erlaubte Adressierungsarten:
nur implizit

Basic-Programm:

```
640  REM ** BRK **
650  PC=PC+2:SP%(SP)=PC/256:SP=SP—1:REM PC+2 AUF STACK
660  SP%(SP)=PC—256*INT(PC/256):SP=SP—1
670  B=1:SR=SROR32:SP%(SP)=SR:SP=SP—1
     :REM STATUS AUF STACK
680  I=1:PC=PEEK (65534)+PEEK (65535)*256
     :REM WEITER BEI INTERRUPT
690  RETURN
```

4/5.4

Ein lauffähiger Assembler in Assembler

Neben dem Thema Assemblerkurs wollen wir bereits im Grundwerk damit beginnen, einen kompletten Assembler, der selbst in Assembler geschrieben ist, darzustellen. Wegen der Länge des Listings werden wir diesen aber - inklusive des Basicladers - auf die ersten Ergänzungsausgaben teilweise verlegen müssen.

Der hier vorgestellte Assembler hat mehrere Funktionen: Einerseits soll er Ihnen einen leistungsfähigen Assembler in die Hand geben, mit dem Sie Ihre Maschinenspracheprobleme zügig bewältigen können. Eines seiner vielen Merkmale ist z.B. die Verlegung in ein externes EPROM, so daß Ihnen fast der gesamte Speicherbereich des 64er zur Verfügung steht. Die benötigten Betriebssystem-Routinen werden dann ebenfalls ausgelagert. Die Vorgehensweise dazu werden wir in einem der Ergänzungsteile besprechen.

Andererseits soll der hier vorgestellte Assembler bei allen in diesem Buch vorgestellten Maschinenspracheprogrammen verwendet werden, um Anpassungsprobleme zu den verschiedenen auf dem Markt befindlichen Assemblern zu vermeiden. Wir können unseren Lesern nicht vorschreiben, welchen Assembler sie sich zu kaufen haben. Der hier vorgestellte Assembler ist für diejenigen, die sich das Eintippen ersparen wollen, auch auf Diskette lieferbar.

4/5.4.1

Allgemeine Merkmale des Assemblers

Der Assembler wurde auf dem Commodore 64 entwickelt, und nutzt daher dessen Möglichkeiten weitgehend aus. Da der C 128 ebenfalls die Programme des C 64 verarbeiten kann, und außerdem eine ähnliche CPU im 128er-Modus benutzt (und somit auch die gleiche Assemblersprache versteht), kann der Assembler auch auf

dem C 128 genutzt werden. Allerdings wird der Assembler dann den Möglichkeiten, die der C 128 bietet, nicht gerecht.

Inzwischen gibt es eine ganze Reihe von Assemblern für die Commodore-Computer, die sich jedoch in Bezug auf ihr Konzept und auch im Befehlssatz unterscheiden. Um hier eine Orientierungshilfe zu bieten, soll hier das Konzept unseres Assemblers beschrieben werden.

4/5.4.1.1

Speicherkonzept

Ein Problem bei der Erstellung eines Assemblers ist es immer, in welchem Speicherbereich der Assembler selbst stehen soll. Hier kann man nur Kompromisse eingehen, da sich irgendwann der Bereich des Assemblers mit dem des zu erstellenden Programmes überschneiden wird. Aus diesem Grund gibt es unseren Assembler in zwei Versionen:

Eine RAM-Vsion, die im Speicher ab $0801 steht, und wie ein Basic-Programm geladen und gestartet wird, sowie

eine Modulversion, die in EPROM's gebrannt ab $8000 im Speicher bereit steht.

Beide Versionen werden mit dem gleichen Quelltext generiert. Da Sie über den Quelltext des Assemblers verfügen, können Sie die Startadresse des Assembler natürlich auch selbst bestimmen, indem Sie die Festlegung im Programmtext ändern, und dann das Programm neu übersetzen.

Das zweite Problem stellt beim C 64 der Speicherbereich von 64 KByte dar. Wie Sie wissen, verfügt der C 64 über 64 KByte RAM, von denen etwa 2 KByte für Betriebssystemvariablen, Puffer und Bildschirm abgerechnet werden müssen. Bleibt ein Speicherbereich von 62 KByte übrig, der unter normalen Umständen praktisch nicht genutzt werden kann.

Hier zeigt das von Commodore gelieferte Basic starke Schwächen. Unser Assembler kann jedoch in der Modulversion den übrigen 62 KByte Speicherbereich nutzen, da ein Modul keinen RAM-Speicher belegt. Dabei können Sie im Betrieb dem Assembler angeben, in welchen Bereichen er Quelltext, Symbole und Programm speichern soll.

Die Speicherbereiche dürfen nur nicht innerhalb des von Betriebssystemen genutzten Bereichs von $0000 bis $0801 liegen, und sich natürlich auch nicht überschnei-

den. Die RAM-Version ist prinzipiell ebenfalls in der Lage, den Speicherbereich von 62 KByte zu nutzen, jedoch liegt hier der Assembler selbst ab $0801, und darf natürlich nicht überschrieben werden, da sonst das Programm abstürzen würde.

Wie es möglich ist, den Speicher so zu verwalten, können Sie in den Kapiteln 2/2.1.6 ‚PLA' und im Kapitel 4/5.4.4.2 ‚Textfile-Aufbau' nachlesen. Hier sei nur gesagt, daß dies nur durch ständiges Umschalten der Speicherverwaltung des Computers mit Hilfe von kurzen, immer im RAM gehaltenen Routinen erreicht werden kann.

Zu dem Speicherkonzept eines Assemblers gehört auch die Behandlung der Massenspeicher. Im Fall des C 64 sind dies der Kassettenrecorder und/oder das Floppy-Laufwerk. Ein arbeiten mit dem Kassettenrecorder ist nicht sinnvoll, da er das einlesen von Quelltexten direkt beim Assemblieren kaum ermöglicht. Eine Zusammenarbeit von Assembler und Kassettenrecorder ist daher nicht vorgesehen.

Die Diskettenstation wird hingegen voll unterstützt. Sie wird als Erweiterung des Arbeitsspeichers betrachtet. Daher können Quelltexte beim assemblieren ausschließlich auf Diskette stehen, und müssen nur zum Editieren (bearbeiten) in den Speicher geladen werden. Der vom Assembler erzeugte Code kann ebenfalls direkt auf Diskette geschrieben werden. Nur die verwendeten Symbole müssen im Arbeitsspeicher gehalten werden. Es wäre wegen der Zugriffszeit und dem Aufwand nicht sinnvoll, diese ebenfalls auf Diskette verwalten zu wollen.

Mit diesem Speicherkonzept ist es daher möglich, sowohl Programmentwicklungen schnell im Arbeitsspeicher zu betreiben, als auch lange Programme auf Diskette zu verwalten. Ein Beispiel hierzu ist der Assembler selbst, dessen Quelltext bereits die Speicherkapazität des C 64 weit übersteigt.

4/5.4.1.2

Behandlung von Symbolen

Ein weiteres wichtiges Merkmal ist die Behandlung von Symbolen. Aus der Art, wie sie berechnet werden und welchen Einschränkungen sie unterliegen, ergeben sich einige Leistungsmerkmale eines Assemblers. Symbole werden in unseren Assembler bis zu zehn Zeichen korrekt formatiert und dürfen darüber hinaus bis zu zwanzig Zeichen lang sein. Groß- und Kleinschrift wird dabei unterschieden. Dies ist vor allem bei recht langen Programmen nützlich. So kann man zum Beispiel alle Symbole klein schreiben, aber jedem Symbol einen Großbuchstaben, der das jeweilige Programmstück kennzeichnet, voranstellen. So werden doppelte Sym-

bole vermieden (z.B. ‚Hloop‘ und ‚Aloop‘), und sofort die Zugehörigkeit zu einem Programmteil erkannt. Die ist aber eine Frage des persönlichen Programmierstils.

Unser Assembler legt Symbole grundsätzlich nur im ersten Durchlauf an. Dies gilt für jede Art von Symbolen, also für Sprungmarken im Programm, wie für durch ‚eq‘ oder ‚in‘ definierte Konstanten. Hieraus ergibt sich bereits eine Einschränkung des Assembler. Die bei ‚eq‘ angegebenen Werte müssen bereits definiert sein, und dürfen sich nicht auf ein nachfolgendes Symbol beziehen. Dieses Problem kann jedoch immer durch die Wahl der Position der Zuweisungszeile im Quelltext umgangen werden. In anderen Assembler werden Konstanten häufig in jedem Durchgang neu berechnet. Wollen Sie ein derartiges Programm übertragen, so müssen sie evtl. einige Zeilen umstellen.

Bsp.:	verboten ist:	korrekt ist:
	symbol: „eq loop+3	nop
	nop	loop: dex
	loop: dex	symbol: „eq loop+3

Wie viele Assembler, unterscheidet auch unserer zwischen verschiedenen Symboltypen. Es gibt Sprungziele, die im Labelfile mit ‚p‘ gekennzeichnet werden, und die Konstanten, die mit ‚f‘ gekennzeichnet werden. Als besondere Arbeitserleichterung prüft der Assembler während des Assemblierens, ob wir ein Symbol nicht nur definieren, sondern auch irgendwo benutzen. Ist dies nicht der Fall, wird das Symbol im Labelfile durch ‚u‘ gekennzeichnet. Diese Kennzeichnungen haben aber keine Auswirkungen auf Eigenschaften oder Gebrauch von Symbolen. Alle Symbole werden vom Assembler gleich behandelt. Sie können an jeder Stelle, an der ein Wert erwartet wird, eingesetzt, und durch ‚+‘, ‚-‘, ‚*‘ oder ‚/‘ mit anderen Symbolen, oder Werten, verkettet werden. Dabei werden Ausdrücke von links nach rechts ausgewertet. Daher ergibt der Ausdruck ‚5+7*3‘ den Wert (5+7)*3=36, und nicht 5+(7*3) =26, was mathematisch exakt wäre. Diesen Umstand weisen aber auch einige Taschenrechner auf.

4/5.4.1.3

Behandlung von Zero-Page-Variablen

Eine weitere Klasse von Symbolen stellen die Zero-Page-Symbole dar. Sie werden zwar nicht gesondert gekennzeichnet, haben aber durch die kürzere Adressierungsart (siehe Assemblerkurs) eine besondere Bedeutung. Unser Assembler erkennt diese Adressierungsart automatisch daran, daß Zero-Page-Variablen einen Wert

kleiner als $100 haben. In anderen Assemblern müssen diese Symbole manchmal gekennzeichnet werden. Dies geschieht dann z.B. durch einen vorangestellten Stern (also z.B.: ‚*wert' statt ‚wert'). Unsere Art die Zero-Page Variablen zu erkennen, kann aber nur korrekt erfolgen, wenn Zero-Page-Symbole vor ihrem Gebrauch definiert werden. Im folgenden Beispiel kann die Zero-Page-Variable erst im zweiten Durchgang erkannt werden:

```
warte:     lda variable
           bne warte
variable:  .eq $34
test:      rts
```

Da der Assembler im ersten Durchgang die Variable ‚variable' noch nicht kennt, reserviert er einen Speicherplatz von zwei Byte für die Variable. Im zweiten Durchgang wird die Variable jedoch nur ein Byte belegen, so daß die im ersten Durchgang berechneten nachfolgenden Symbole (hier: ‚test') nicht mehr stimmen. Dieser Fehler wird am Ende des zweiten Durchgangs zwar vom Assembler erkannt und gemeldet (‚Berechnungs-Fehler'), aber der Assembler kann nicht feststellen, in welcher Zeile der Fehler gemacht wurde. Sie sollten sich daher angewöhnen, als erstes immer alle Symbole zu definieren. Dies ist sogar generell zu empfehlen, da es der Übersichtlichkeit und Lesbarkeit eines Programmes dient.

4/5.4.1.4

Abweichungen von anderen Assemblern

Weitere Abweichungen von anderen Assemblern betreffen den Satz der Pseudo-Befehle der Assembler. Dies sind Befehle, die zur Steuerung des Assemblers, zum direkten Ablegen von Werte im Programm, und der Ausgabesteuerung dienen. Es gibt hier zwar so etwas wie einen Standard, dieser ist aber so gering, daß praktisch alle Assembler einen eigenen Satz an Pseudo-Befehlen haben. Unser Assembler ist hier keine Ausnahme. Im Folgenden werden daher Angaben zum Umsetzen anderer Assemblerprogramme gemacht. Diese sind jedoch keinesfalls vollständig! Der volle Befehlssatz unseres Assemblers ist im Kapitel 4/5.4.2.6 ‚Pseudo-Befehle' mit den zugehörigen Erklärungen aufgeführt.

Festlegung der Basisadresse

Unser Assembler legt die Basisadresse mit dem ‚.ba'-Befehl fest. In anderen Assemblern wird ‚*=', ‚org' oder ‚.org' verwendet.

Definieren von Symbolen

Symbole werden durch ‚.eg' festgelegt oder durch ‚.in' eingelesen. In anderen Assembler heißt der ‚.eq'-Befehl häufig ‚equ', ‚.equ', ‚.de' oder ‚='.

Ablegen von Werten

Werte können durch ‚.by', ‚.wo', ‚.ad', ‚.dp' oder ‚rv' abgelegt werden. In anderen Assembler heißt ‚.by' z.B. ‚.byte', ‚.byt', oder ‚byte'. ‚.wo' heißt entsprechend ‚.word' oder ‚word'. Die Befehle ‚.dp' und ‚.rv' findet man im allgemeinen nicht in anderen Assemblern.

Reservieren von Speicherbereichen

Speicherbereiche werden mit ‚.db wert' oder ‚.dw wert' definiert. In anderen Assembler wird dies durch ‚*=*+ wert' oder ‚.ds' erreicht.

Beispiel: ‚. db 128' wird zu ‚*=*+128'.

Quelltextende

Das Ende des Quelltextes wird fast immer automatisch erkannt, kann aber auch durch ‚.en', ‚.end' oder ‚end' erreicht werden.

Bedingte Assemblierung

Bei den Befehlen zur bedingten Assemblierung (‚.?p', ‚?m', ‚.?e', ‚.?n', ‚.eb') gibt es viele Möglichkeiten. Sie sind aber selten anzutreffen. Im allgemeinen haben sie Namen der Form ‚ifp', ‚ifm', ‚ife', ‚ifn' oder den Anweisungen ‚if', ‚else' und ‚endif'.

Formatsteuerung

Die Formatsteuerung wird ebenfall recht unterschiedlich gelöst, ist aber für das Umsetzen von Programmen auch nicht erforderlich.

Mit Hilfe dieser Angaben sollten Sie in der Lage sein, Programme von anderen Systemen zu übertragen. Häufig erkennt man unbekannte Befehle auch aus den Kommentaren oder der Programmbeschreibung.

4/5.4.2

Bedienungsanleitung

Im Folgenden werden alle Möglichkeiten des Assemblersystems aufgezeigt. Damit Sie richtig mit dem System arbeiten können, werden alle Befehle getrennt nach Gruppen vorgestellt. Sie können daher in den entsprechenden Kapiteln die Befehle nachschlagen.

4/5.4.2.1

Laden und Starten

Der Assembler ASS wird mit LOAD „ass.ram“, 8 von der Diskette geladen und mit RUN gestartet. Er meldet sich dann mit seinem Menü.

In dem Menü stehen neben den Menübefehlen, die Anzahl der Zeilen, die der Text im Textspeicher enthält, ob der Drucker aktiv geschaltet ist und die Grenzen des Text- und Symbolspeichers. So bedeutet ‚Text: $8000-$DFFF gefüllt: $81E4‘ z.B. das der Textspeicher von $8000 bis $DFFF liegt, und bis $81E4 gefüllt ist.

4/5.4.2.2

Die Menü-Befehle

Nach dem Starten befinden Sie sich im Hauptmenü von ASS. Von hier werden die wichtigsten Grundfunktionen erreicht und es erfolgt die Verzweigung in die Programmmodule Assembler, Editor oder Monitor.

Alle Befehle werden durch einen Tastendruck gestartet. Benötigt ASS weitere Informationen, so werden diese abgefragt.

Folgende Befehle können vom Menü aus erreicht werden:

Taste	Befehl
E	Sprung in den **E**ditor
A	**A**ssembler starten
P	zweiten **P**ass des Assemblers starten
B	**B**asic starten. Es wird ein Reset ausgelöst, und so Basic gestartet.
W	**W**rite: Der im Textspeicher stehende Text wird auf den Drucker ausgegeben.
O	**O**ptions: Die Speichergrenzen für Text und Symbolspeicher können eingegeben werden. Die momentanen Grenzen werden dabei vorgegeben und können mit <RETURN> bestätigt werden. Zulässig sind Zahlen in dezimaler, binärer (%10) oktaler (&0321) oder sedezimaler ($FFD2) Darstellung. Beachten Sie bitte die vorangestellten Sonderzeichen: ‚%‘ für binär, ‚&‘ für oktal und ‚$‘ für sedezimal.
R	**R**un: Ein Unterprogramm kann gestartet werden. ASS fragt zunächst nach der Startadresse der Routine. Als Adresse kann eine Zahl, ein Label, oder ein Ausdruck eingegeben werden (z.B. Loop+$45). Dann fragt ASS, ob alles auf RAM geschaltet werden soll. Wird die Frage mit ‚J‘ beantwortet, so wird vor dem Starten der gesamte ROM-Bereich abgeschaltet. Sonst verhält sich der Computer wie nach dem Einschalten. Die Routine muß mit RTS enden!
I	**I**nit: ASS wird neu gestartet.
G	**G**et: Eine Textdatei wird in den Textspeicher geladen. Der Dateiname wird von ASS erfragt. Ist die Textdatei zu groß, wird ein Fehler gemeldet. Der Textspeicher muß dann vergrößert werden.
S	**S**ave: Der momentane Text wird auf Diskette gespeichert.
M	**M**erge: An den Text im Speicher wird der Text aus einer Textdatei angehängt.

Taste	Befehl
V	Verify: Der Text im Speicher wird mit einer Textdatei verglichen. Ist der Text verschieden, wird ein Fehler gemeldet. Sonst erfolgt keine Meldung.
C	Catalog: Das Inhaltsverzeichnis der Diskette wird angezeigt. Der Befehl benötigt eine Maske (Disk-Mask). <RETURN> liefert das gesamte Inhaltsverzeichnis. Sollen z.B. alle File's, die mit ‚S' beginnen angezeigt werden, so wird als Maske ‚S*' eingegeben.
D	Disk-Befehl: Sendet einen Diskettenbefehl. Soll z.B. das File ‚TEST' gelöscht werden, so wird ‚S: TEST' eingegeben.
F	Floppy: Liest den Fehlerkanal der Floppy aus.
L	Label-File: Zeigt die definierten Symbole an. Es kann angegeben werden, ob alle Symbole, oder nur die Symbole eines bestimmten Typs ausgegeben werden. Folgende Typen sind möglich: Fix = alle Symbole die mit ‚.eg' oder ‚.in' definiert wurden Program = alle Symbole die als Label im Programmtext vorkommen Unused = alle Symbole die zwar definiert wurden, aber nicht benutzt wurden Ist der Drucker eingeschaltet, so geht die Ausgabe auf Drucker.
T	Toggel: Schaltet den Drucker ein bzw. aus.
H	Help: Erlaubt die Eingabe einer Adresse der Form wie bei R (Run) und gibt diese dezimal, sedezimal (Hex. durch $ gekennzeichnet), oktal (Basis 8, durch & gekennzeichnet) und binär (durch % gekennzeichnet) aus.
N	Sprung in den Monitor.

4/5.4.2.3

Der Editor

Nach dem Einschalten des Editors sehen Sie ein fast leeres Bild. Nur in der oberen Zeile stehen Informationen. Dies ist die Statuszeile. Sie zeigt laufend die Zeilennummer und die Position des Cursors an. Zeilen können max. 80 Zeichen haben. In der Statuszeile wird außerdem noch die aktuelle Betriebsart angezeigt. Etwa in der Form: ‚F:+ P:- I:- Q:+'. Dabei bedeutet ‚+', daß die Betriebsart eingeschaltet, und ‚-' daß sie ausgeschaltet ist.

F:+ bedeutet also, daß die Ausgabe auf Drucker und Bildschirm auf Assemblerformat gebracht wird.
I:+ bedeutet, daß nach jeder Zeile automatisch eine neue erzeugt (Insert-Mode), und jedes Zeichen in eine Zeile eingefügt wird.
P:+ bedeutet, daß gepackt wird, d.h. aus jeder Zeile werden unnütze Leerzeichen gelöscht.
Q:+ bedeutet, daß ein ”"‘ eingegeben wurde. Bis zum nächsten ”"‘ können nun auch Steuerzeichen in den Text übernommen werden.

4/5.4.2.4

Die Editor-Befehle

Im Editor sind folgende Tasten belegt:

F1: Zum Textanfang	F2: Zum Seitenanfang
F3: Eine Seite zurück	
F5: Eine Seite vor	
F6: Zum Textende	F8: Zum Seitenende

Die INS/DEL–Taste funktioniert gemäß ihrer Beschriftung.

Mit der CTRL-Taste kommen Sie in den Befehlmodus.
Folgende Befehle sind dann möglich:

E	–	Zeile löschen
N	–	Zeile einfügen
D	–	Block löschen
C	–	Block kopieren
M	–	Block verschieben
W	–	Block auf Drucker ausgeben
A	–	Blockanfang festlegen
B	–	Blockende festlegen
Z	–	Blockdefinition löschen
L	–	Links vom Cursor löschen
R	–	Rechts vom Cursor löschen
K	–	Zeileninhalt löschen
Y	–	Korrektur löschen
F	–	Suchen
S	–	Ersetzen von Zeichenketten
O	–	Old (Textdatei retten)
G	–	Goto Zeilennummer
T	–	Format on/off
I	–	Insert Mode on/off
P	–	Pack on/off
X	–	Text löschen
Q	–	Zum Menü
<RETURN>	–	Kein Befehl

Bei den Befehlen Suchen und Ersetzen können im Suchwort unbedeutende Stellen mit ‚?‘ gekennzeichnet werden (sogenannte Platzhalter oder „Wild Cards“).

4/5.4.2.5

Aufbau des Quelltextes

Eine Zeile hat folgenden Aufbau:

Symbol: Befehl Operand; Kommentar

Ein **Symbol** ist eine Zeichenkette, die mit einem Buchstaben beginnt, und mit weiteren Buchstaben oder Zahlen, oder den Zeichen ‚.‘,‚?‘ oder ‚!‘ fortgesetzt werden kann. Es darf maximal 10 Zeichen lang sein. Es wird durch einen Doppelpunkt vom Rest der Zeile abgetrennt. Es kann auch fehlen. Groß- und Kleinschrift wird unterschieden.

Ein **Befehl** ist einer der offiziellen 6502 Mnemo-Befehle oder ein Pseudo-Befehl. Er kann auch fehlen.

Ein **Operand** ist ein Argument verbunden mit einer Adressierungsart des 6502. Die Zero-Page-Adressierung wird automatisch erkannt. Der Operand kann auch fehlen. Bei der Zero-Page Adressierung muß der Operand vor dem ersten Aufruf definiert sein!

Ein **Argument** ist eine Zahl oder ein Symbol, oder mehrere Zahlen oder Symbole verbunden durch ‚+‘, ‚-‘, ‚*‘ oder ‚/‘. Es wird von links nach rechts ausgewertet. Wird einem Argument ‚<‘ vorangestellt, so wird der niederwertige Teil angesprochen, bei ‚>‘ der höherwertige Teil. Das heißt, es wird das höherwertige bzw. niederwertige Byte eines 16-Bit-Wertes angesprochen (siehe auch Einführungsteil). Als Zahl kann ebenfalls der ASCII-Wert eines Zeichens angegeben werden (z.B. ‚a‘).

Ein **Kommentar** ist eine beliebige Zeichenkette, die mit einem ‚;‘ beginnt.

Alle **Zahlen** können dezimal (Basis 10, z.B. 121), sedezimal (hexadezimal Basis 16, z.B. $12ff), oktal (Basis 8, z.B. &72) oder binär (Basis 2, z.B. %101) eingegeben werden.

Zulässige Zeilen sind z.B.:

```
lda (loop), y
loop: lda #45+31-20
cmp #‘a
abc: jmp$ffc3; Beispiel 1
cmp #>$fcd3
```

- Soll als Argument der Akkumulator angesprochen werden, so wird nicht, wie bei den meisten Assemblern, z.B. ASL A, sondern nur ASL eingegeben.
- Soll der momentane Programmzähler im Argument verwendet werden, so wird er durch das Symbol ‚*‘ erreicht (z.B. beq*+5). ‚*‘ darf nur in der ersten Position stehen (d.h. beq 5+*‘ ist verboten).

4/5.4.2.6

Pseudo-Befehle

Der Assembler ASS kennt folgende Pseudo-Befehle, die alle mit einem Punkt beginnen:

symbol: .eq operand

Weise dem Symbol folgenden Wert zu.
Bsp.: loop: .eq$d000+3

symbol: .in „text“

Drucke text, hole eine Zahl von der Tastatur und weise sie dem Symbol zu. Der Text kann auch fehlen.

Bsp.: txtanf: .in „textanfang?“
txtend: .in

.oo operand

Der Operand wird sedezimal ausgegeben.

.tx „Text“

Der Text wird ausgegeben.

.en

Beende den Paß.

.db operand

Reserviere soviele Byte, wie der Wert von operand angibt.

.dw operand

Reserviere soviele Wörter (1 Wort = 2 Byte), wie der Wert von operand angibt.

.wa

Bei einer Fehlermeldung wartet der Computer. Durch Drücken der ‚C'-Taste (‚C' für continue) wird der Paß fortgesetzt, und durch Drücken der ‚A'-Taste der Assembliervorgang abgebrochen. Steht der Quelltext im Speicher kann durch Drücken der ‚E'-Taste in den Editor zu der fehlerhaften Zeile gesprungen werden.

.co

Hebe .wa auf

.be operand

Der Wert von operand darf zwischen Null und Zwei liegen. Ist der Operand Null, wird in jedem Paß bei Erreichen des Befehls ein Ton ausgegeben. Ist der Operand ungleich Null nur im jeweiligen Paß.

.?e operand

Ist der Operand Null, so wird der Block übersetzt, anderenfalls ignoriert und die Assemblierung nach dem .eb Befehl fortgesetzt.

.?n operand

Ist der Operand ungleich Null, so wird der Block übersetzt, anderenfalls ignoriert.

.?p operand

Ist der Operand positiv, so wird der Block übersetzt, anderenfalls ignoriert.

.?m operand

Ist der Operand negativ, so wird der Block übersetzt, anderenfalls ignoriert.

.?1

Der Block wird nur in Paß 1 übersetzt.

.?2

Der Block wird nur in Paß 2 übersetzt.

.eb

Ende des Blocks.

Beispiel:

```
ver.: .in „version 1 oder 2?“
      .?e ver-1
10:   lda #1 ;block 1
      sta $50
      .eb
      .?e ver-2
10:   dex    ;block 2
      .eb
```

Die Blöcke dürfen nicht verschachtelt werden!

.os

Lege im 2. Paß den Code ins RAM ab.

.oc

Hebe .os auf.

.ba operand

Weise dem Programmzähler den Wert von operand zu.

.bc operand

Lege den erzeugten Code, statt bei der durch .ba festgelegten Adresse, bei der Adresse operand ab. Eine .ba Anweisung hebt eine .bc Anweisung auf. .os muß extra gegeben werden.

.cb operand

Weise dem Programmzähler den Wert zu, ohne die .bc Anweisung aufzuheben.

.ls

Druck-Protokoll.

.lc

Hebe .ls auf.

.ou „file“

Schreibe den erzeugten Code in das File mit dem angegebenen Namen und übergebe als Startadresse den momentanen Programmzähler.

.fi „file“

Setze die Assemblierung im File mit dem angegebenen Namen fort. Dieser Befehl geht nur bei Assemblierung von Diskette.

.by opkette

Lege die Operanden byteweise ab.

.wo opkette

Lege die Operanden wortweise ab. (Reihenfolge: <operand, >operand nach 6502 Standard)

.ad opkette

Wie, Wo, jedoch mit Reihenfolge >operand, <operand.

.rv opkette

Wie .by, jedoch wird rückwärts abgelegt. D.h. .rv 2+3,1 gibt $01 $05.

.dp „text“

Lege den Text als Display-Code ab. Dabei wird der ASCII-Code in den Code umgerechnet, der in der POKE-Tabelle steht. Bsp.: .dp „aA“ ergibt $01 $41.

Eine **opkette** ist ein oder mehrere Operanden, getrennt durch „,‘ wobei als Operand auch eine Zeichenkette eingeschlossen in ‚“‘ angegeben werden kann.
Bsp.: .by 51-45, $4f, ‚a,%100, „probe text“, <$d2f4,>$ff66.

4/5.4.2.7

Fehlermeldungen

Außer den Fehlermeldungen die ASS liefert, kann das Betriebssystem des C64 zehn Fehlermeldungen der Form „I/O ERROR #?“ liefern, wobei ‚?‘ die Nummer des Fehlers ist. Dabei bedeutet:

0	Stop-Taste gedrückt		
1	Too many files	–	Zu viele Files eröffnet
2	File open	–	File ist bereits eröffnet
3	File not open	–	File ist nicht eröffnet
4	File not found	–	File wurde nicht gefunden
5	Device not present	–	Gerät nicht korrekt angeschlossen
6	Not input file	–	Aus dem File kann man nicht lesen
7	Not output file	–	Auf das File darf man nicht schreiben
8	Missing file name	–	Filename fehlt
9	Illegal device number	–	Gerätenummer ist verboten

Diese Fehlermeldungen treten i.a. jedoch nicht auf!

ASS liefert folgende Fehlermeldungen:

Sprung zu lang

Ein bedingter Sprung (z.B. BNE) ist weiter, als die CPU erlaubt.

Operand

Der angegebene Operand kann von ASS nicht entziffert werden.

Symbolanzahl

Es werden mehr Symbole angelegt, als in den Symbolspeicher passen. Die Assemblierung wird von ASS abgebrochen!

Adr. unbekannt

Es wurde eine Adressierungsart angegeben, die die CPU nicht kennt [z.B.: LDA ($34), Z].

Adr. nicht möglich

Die angegebene Adressierungsart kann bei dem angegebenen Befehl nicht verwendet werden. (z.B.: STX adresse, X).

Symbol unbekannt

Das im Operand angegebene Symbol wurde nicht definiert.

Befehl unbekannt

Der angegebene Befehl ist weder ein 6502-Befehl noch ein Pseudo-Befehl.

Operand zu groß

Der angegebene Operand ist zu groß. Beispiel: LDA #300.

Verschachtelt

Bei der Assemblierung eines Block's wurde in dem Block ein neuer Block geöffnet.

Symbol fehlt

Bei einem .eq oder .in Befehl wurde kein Symbol angegeben.

Block offen

Nachdem ein Block geöffnet wurde, folgt bis zum Fileende kein .eb Befehl.

Symbol doppelt

Das Symbol wurde bereits definiert.

Basis fehlt

Für das Programm wurde keine Basis (mit .ba) festgelegt.

Befehl doppelt

Ein .ou Befehl wurde zweimal gegeben.

Mode

Ein .fi Befehl wurde gegeben, obwohl nicht im Disk-Modus gearbeitet wird.

Berechnung

Eine Zero-Page-Variable wurde nach ihrem ersten Aufruf definiert. Die Symbolberechnungen aus Paß 1 stimmen daher mit denen aus Paß 2 nicht mehr überein!

Dieser Fehler wird erst am Ende der Assemblierung erkannt. Die angegebene Zeilennummer hat daher keine Bedeutung!

4/5.4.2.8

Assemblieren

ASS kennt zwei Möglichkeiten einen Text zu assemblieren: Zum einen kann der Text im Speicher des C 64 stehen, und zum anderen kann sich der Text in einer Datei auf Diskette befinden.

Um einen Text zu assemblieren, muß zunächst die Taste ‚A‘ gedrückt werden. ASS fragt dann, ob er ein Protokoll des Vorgangs drucken soll. Diese Frage wird durch Drücken der Taste ‚J‘ bzw. ‚N‘ entsprechend ‚Ja‘ und ‚Nein‘ beantwortet. Ist der Drucker eingeschaltet, so wird das Protokoll auf dem Drucker geleitet.

Dann fragt ASS nach einem Filenamen. Beantworten Sie diese Frage, so übersetzt ASS die Textdatei auf Diskette, ohne den Textspeicher in Anspruch zu nehmen. Beantworten Sie die Frage mit <RETURN>, so wird die Datei im Speicher übersetzt.

Der zweite Paß kann auch direkt gestartet werden.

4/5.4.2.9

Speicherbereiche

ASS belegt außer dem Programmbereich eine Reihe von Speicherstellen in der Zero-Page, sowie alle Pufferbereiche im Bereich \$0000–\$0400 (d.h. Basic-Eingabepuffer, Kassettenpuffer etc). Ein mit dem RUN-Befehl gestartetes Programm darf diese Bereiche mit Ausnahme von 679-767 benutzen. In diesen Bereich rettet ASS alle wichtigen Daten. Nach dem Programmlauf werden die Speicherstellen sofort wieder von ASS belegt.

4/5.4.2.10

Monitor

ASS bietet außerdem einen kleinen Monitor. Bei den Monitorbefehlen müssen alle Adressen als vierstellige Hexadezimalzahlen eingegeben, und mit einem Leerzeichen getrennt werden. Folgende Befehle sind möglich:

d <von> <bis>

Disassemblieren ab Adresse <von> bis Adresse <bis>. <bis> kann auch entfallen.

f <von> <bis> <operanden>

Suchen der Bytefolge <operanden> im Speicherbereich <von> bis <bis>.

Bsp.:
f A000 AFFF A9 „test“ B7 sucht im Bereich $A000-$AFFF nach der Bytefolge A9 „test“ B7. Wird die Kombination gefunden, so wird ihre Adresse ausgegeben und weitergesucht.

g <adresse>

Ruft das Programm ab <adresse> auf (JMP adresse).

u <adresse>

Ruft das Unterprogramm ab <adresse> auf (JSR adresse).

l <adresse> „name“

Das Programm „name“ wird von Diskette gelesen und ab der Adresse <adresse> gespeichert. Fehlt <adresse>, so wird das Programm an die Originaladresse geladen.

s „name“ <von> <bis>

Speichert den Bereich <von> bis <bis> unter dem Namen „name“ auf Diskette.

m <von> <bis>

Erstellen eines Hexdumps im Bereich <von> bis <bis>. <bis> kann auch entfallen. Die Daten können überschrieben werden.

= <adr1> <adr2>

Vergleicht die Bereiche ab <adr1> bzw. <adr2> miteinander, und gibt die Adresse des ersten abweichenden Bytes aus.

p+/-

p+ schaltet den Drucker an, p- schaltet ihn aus.

a <von> <bis>

Ausgabe des Bereichs <von> bis <bis> als ASCII Zeichen. <bis> kann auch entfallen. Die Daten können überschrieben werden.

c <von> <bis>

Ausgabe des Bereichs <von> bis <bis> als POKE-Code. <bis> kann auch fehlen. Die Daten können überschrieben werden.

v <von> <bis> <nach>

Verschieben des Bereichs <von> bis <bis> in den Bereich mit der Startadresse <nach>.

b <track> <sector>

Lese den Block <track> <sector> von Diskette.

b

Lese den logisch nächsten Block.

w <track> <sector>

Schreibe den Block als Block <track> <sector> auf Diskette.

w

Schreibe den Block in den zuletzt angesprochenen Block.

i

Gebe Track und Sector des aktuellen Block's aus.

e <von> <bis>

Gebe die Bytes <von> bis <bis> des aktuellen Blocks aus. <bis> kann auch fehlen. Die Daten können überschrieben werden. <von> und <bis> sind als zweistellige Hexadezimalzahlen anzugeben.

e

Liest den Fehlerkanal der Floppy aus.

r

Gibt die Registerinhalte aus.

Bsp.:

pc	ac	xr	yr	sp	nv-dbizc	chl	sr	pp
c000	01	02	03	f4	11101111	111	ef	37

Dies bedeutet:
der Programmzähler steht auf $C000 (pc = c000)
der Akku enthält 1 (ac = 01)
das X-Register enthält 2 (xr = 02)
das Y-Register enthält 3 (yr = 03)
der Stackpointer enthält $F4 (sp = f4)
Die Flaggen sind gemäß des 0/1 Feldes gesetzt
Die internen Leitungen char/hiram/lowram sind gemäß der Bitfolge gesetzt.

Alle diese Daten können verändert werden. Nicht überschrieben werden können die Zusatzinformationen des Statusregisters (sr = ef) und des Prozessorports (pp = 37; Adr. 1).

4/5.4.2.11

Sonstiges

- Alle Funktionen sind auf den gesamten Speicherbereich des C 64 anwendbar also auch Load/Save im RAM unter I/O und ROM)
- im Monitor bestimmt die Lowram/Hiram Angabe, ob die Funktionen auf RAM/ ROM oder I/O angewendet werden.
- Die meisten Funktionen lassen sich mit der STOP-Taste abbrechen.
- Die Ausgabe kann durch Drücken der <SHIFT>-Taste angehalten, und mit der <C=>-Taste fortgesetzt werden.

- Der weitverbreitete Programmiertrick

```
eins: lda # 1
      .by $2c
zwei: lda # 2
      sta speicher
```

kann als

```
eins: lda # 1
      bit
zwei: lda # 2
      sta speicher
```

geschrieben werden. ‚bit' entspricht als ‚by $2c'.

4/5.4.3

Listing

Das Programmm befindet sich auf der Grundwerksdiskette

4/5.4.3.1

Hinweise zur Eingabe und Assemblierung

Das Listing des Assemblers ist selber ein Assemblerlisting. Sie können es daher nicht einfach eintippen und mit ‚RUN‘ starten, sondern müssen es erst in ein lauffähiges Programm wandeln. Ihnen stehen dazu mehrere Möglichkeiten offen.

Falls Sie bereits im Besitz eines Assemblers sind, können Sie mit seiner Hilfe das Listing übersetzen. Dabei ist zu beachten, daß der Quelltext nicht in den Speicher des C64 paßt. Er wurde daher in mehrere Teile geteilt, die durch den ‚.fi‘-Befehl nachgeladen werden. Der Name einer Quelldatei ist in einer Kommentarzeile jeweils angegeben. Sollten Sie von dieser Möglichkeit Gebrauch machen, müssen Sie wahrscheinlich auch einige der Pseudo-Befehle ändern, da die verschiedenen Assembler hier unterschiedliche Eingaben erwarten.

Eine weitere Möglichkeit besteht darin, den Assembler mit Hilfe des in diesem Buch vorgestellten Basic-Assemblers zu übersetzen. Der Basic-Assembler bietet keine Schwierigkeiten bei der Eingabe, da er alle verwendeten Befehle versteht. Aber auch hier muß der Quelltext natürlich wieder in verschiedenen Dateien abgelegt werden.

Wenn Sie eine dieser Möglichkeiten nutzen, so werden Sie während des Assemblierens vom Assembler nach verschiedenen Eingaben gefragt. Als erstes erscheint die Frage, ob eine RAM oder eine Modulversion erzeugt werden soll. In der RAM-Version wird ein von Basic aus mit ‚RUN‘ startbares Programm erzeugt. Die Modulversion steht hingegen ab $8000 im Speicher und muß in Eprom’s gebrannt werden, um dann als Modul in den vorgesehenen Platz gesteckt zu werden. Dann werden Sie gefragt, ob die Ausgabe auf Diskette erfolgen soll. Beantworten Sie diese Frage mit ‚Nein‘, wird der Quelltext nur zur Probe assem-

bliert, d.h. es erfolgt keine Ausgabe des erzeugten Programms. Als letztes werden Sie gefragt, ob Standardwerte verwendet werden sollen. Beantworten Sie diese Frage mit ‚Nein', so werden Sie nach Text- und Symbolspeicherbereich, Textfarben, Drucker und Diskettenparameter gefragt. Die jeweiligen Standardwerte werden vorgegeben und können mit ‚RETURN' bestätigt werden.

Die letzte Möglichkeit besteht in der Eingabe des Assemblers mit Hilfe des Basic-Laders. Der Basic-Lader ist ein Basicprogramm, in dem das übersetzte Programm bereits abgelegt ist. Es generiert aus diesen Daten dann ein lauffähiges Programm auf Diskette. Dabei werden gleichzeitig die Daten überprüft, so daß Fehler weitgehend ausgeschlossen sind.

Jede dieser Möglichkeiten hat Vor- und Nachteile. Der Basic-Lader ist sicherlich die bequemste und sicherste Möglichkeit der Eingabe, aber Ihnen steht hierbei nicht der Quelltext zur Verfügung. Soll also etwas verändert, korrigiert oder verbessert werden, ist dies nur schwer zu erreichen. Erweiterungen, die in diesem Buch vorgestellt werden, werden natürlich den Basic-Lader berücksichtigen. Bei eigenen Änderungen sind Sie aber auf sich gestellt.

Bei den beiden anderen Möglichkeiten steht Ihnen der Quelltext zur Verfügung. Sie können daher leicht im Programm etwas ändern. Dafür muß die Eingabe aber sehr sorgfältig geschehen, da eine fehlende Zeile bereits zum Programmabsturz führen kann.

Für welche der Möglichkeiten Sie sich entscheiden, liegt also bei Ihnen. Auf jeden Fall sollten Sie jedoch vor dem Starten des Programms dieses speichern! Außerdem sollten Sie die verschiedenen Funktionen des Programms testen, bevor Sie die Arbeit des Programmierens beginnen. So bleibt Ihnen vielleicht der Absturz einer längeren Textdatei erspart. Sollte beim Testen eines von Ihnen geschriebenen Programms der Computer doch einmal abstürzen, so können Sie den Text häufig retten, wenn Sie nach einem RESET den Assembler neu laden, in den Editor gehen und den Befehl ‚O' (OLD) eingeben.

4/5.4.4

Programmbeschreibung

In diesem Kapitel wird eine Übersicht über das Programm des Assemblers gegeben. Natürlich ist es aufgrund des Umfangs des Programms nicht möglich, auf die Arbeitsweise jeder Routine einzugehen. Dies müssen Sie anhand des Listings selbst durchführen. Es soll hier im wesentlichen der Programmaufbau und die Struktur der verwendeten Daten erklärt werden.

Um den Überblick über das Programm zu behalten, wurde das Listing in Abschnitte eingeteilt, denen ein Abschnitt-Kopf vorangestellt wurde. Dieser Kopf ist mit Sternchen (‚*') umrahmt, so daß sie ihn leicht finden. Ist also im Folgenden vom Abschnitt ‚Editor' die Rede, so ist der Programmteil nach dem Kopf ‚Editor' gemeint. Wird in einem Abschnitt auf eine bestimmte Routine verwiesen (z.B.: <move>), so wird sie mit dem Namen bezeichnet, mit dem sie aufgerufen wird. Um Abschnitt-Köpfe in einer Textdatei leichter zu finden, können Sie mit dem Editor-Befehl ‚F' nach der Zeichenfolge ‚;**' suchen.

4/5.4.4.1

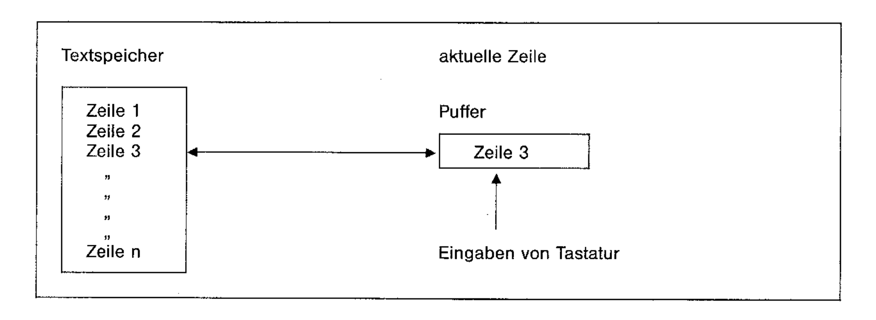
Arbeitsweise des Editors

Um mit einem Assembler arbeiten zu können, benötigt man einen Editor, mit dem man die Textdateien erstellt. Im einfachsten Fall ist dies der Basic-Editor. Der Basic-Editor des C 64 ist aber nicht besonders komfortabel. Daher wurde für den Assembler ein eigener Editor erstellt. Sie finden den Editor im gleichnamigen Abschnitt.

Der Editor arbeitet zeilenorientiert, d.h. intern werden Zeilen verarbeitet, wie bei dem Basic-Editor auch. Alle Editor-Befehle arbeiten daher nur auf Zeilen. So ist z.B. das Definieren eines Blockes nicht mitten in einer Zeile, sondern nur am Anfang der Zeile möglich. Die Zeilen werden jedoch nicht über Zeilennummern angesprochen, sondern über ihre Position auf dem Bildschirm. Dazu merkt sich

das Programm die Nummer der ersten Zeile auf dem Bildschirm in der Speicherzeile ‚bildzeile'. Aus diesem Wert und der Zeile auf dem Bildschirm kann so die Zeilennummer berechnet werden.

Wird nun eine Taste gedrückt, so wird getestet, ob es sich um die Control-Taste handelt, und gegebenenfalls die Kommandoroutine aufgerufen. Wird dagegen ein Zeichen eingegeben, so muß die momentane Zeile geändert werden. Zu diesem Zweck wird sie zuerst in den internen Puffer ‚edbuffer' kopiert, in dem die Zeile dann geändert werden kann. Sämtliche Änderungen werden nun in dem Puffer ausgeführt. Verläßt man die Zeile mit der RETURN- oder einer Cursor-Taste, so wird die Zeile im Text aktualisiert (<updaten>). Der Einfachheit halber wird dazu erst die Zeile im Text gelöscht und dann die neue Zeile eingefügt. Auf den beiden Grundroutinen des Löschen und Einfügens basieren alle Befehle des Editors. Sie finden die Routinen <delzeile> und <inszeile> im Abschnitt ‚Text bearbeiten'. Sie werden in der Routine <change> noch einmal zusammengefaßt.

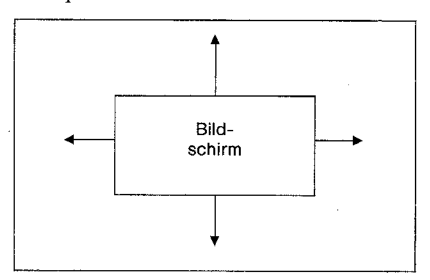


Die beiden Grundroutinen basieren wiederum auf der Routine <move> im Abschnit ‚Move'. Diese Routine verschiebt Speicherbereiche. Dazu wird der Speicherbereich und die neue Adresse angegeben. Daraus berechnet <move> die Verschieberichtung und verschiebt den Bereich. Mit dieser Routine lassen sich so nicht nur Zeilen löschen und einfügen, sondern ganze Textblöcke verschieben. Dies ist nur wenig komplizierter. Ein Block muß zunächst markiert werden. Zu diesem Zweck merkt sich das Programm die Zeilennummern bei den Editor-Befehlen ‚A' und ‚B' in den Speicherstellen ‚bloanf' bzw. ‚bloend'. Damit ist ein Block definiert. Soll der Block verschoben werden, so werden aus den Blockzeilen die Adresse des Blocks berechnet, und aus der Cursorposition die Zieladresse. Dann wird mit <move> der Block an die Adresse kopiert und anschließend der Originalblock wiederum mit <move> gelöscht.

Die Routinen mit den Editor-Befehlen finden Sie im Abschnitt ‚Editor-Befehle' und, soweit es Blockbefehle sind, im Abschnitt ‚Block-Befehle'. Sie sind recht einfach zu verstehen, da sie aus mächtigeren Routinen (wie z.B. <move>) zusammengesetzt sind. Besonders erwähnenswert sind im Abschnitt ‚Editor-Befehle' noch die Routinen <bloinskorr> und <blodelkorr>. Sie korrigieren die vom Benutzer festgelegten Blockgrenzen, falls in dem Block Zeilen eingefügt (<bloinskorr>) oder gelöscht werden (<blodelkorr>).

Wie bereits erwähnt, ermittelt unser Editor die Zeilennummern aus ihrer Position auf dem Bildschirm. Dazu müssen die Zeilen zunächst auf dem Bildschirm dargestellt werden. Auch hierzu gibt es eine Reihe von Standardroutinen. So finden Sie im Abschnitt ‚Statuszeile drucken' die Routine <stazei>, die die oberste Zeile einer Seite mit den Informationen über Position im Text und aktive Funktionen beschreibt. Die Routine <bild> im Abschnitt ‚Eine Seite anzeigen' füllt hingegen eine Bildschirmseite mit Zeilen. Sie wird beim Editorstart und horizontalem Scrollen benötigt. Da eine Zeile bis zu 80 Zeichen beinhalten kann, aber nur 40 auf den Bildschirm passen, wird der Bildschirm als Fenster über den Text verwaltet. Die Routine <bild> holt sich daher von <fianzpos> (Abschnitt ‚Editor') die Position des linken Randes des Textfensters. Um nun eine Zeile anzuzeigen, muß ermittelt werden, ob die Zeile gerade bearbeitet wird. Dann wird die Aufgabe an <edpri> (Abschnitt ‚Editor') weitergeleitet. Sonst ist die Zeile im Textspeicher zu finden und wird von <bild> angezeigt. Dabei muß <bild> die Zeile im allgemeinen noch auf das richtige Format bringen (falls eingeschaltet). Die nötigen Positionen von Label und Kommentar liefert hierzu die Routine <instr> im Abschnitt ‚Editor', die ein Zeichen in der Zeile sucht.

Textspeicher

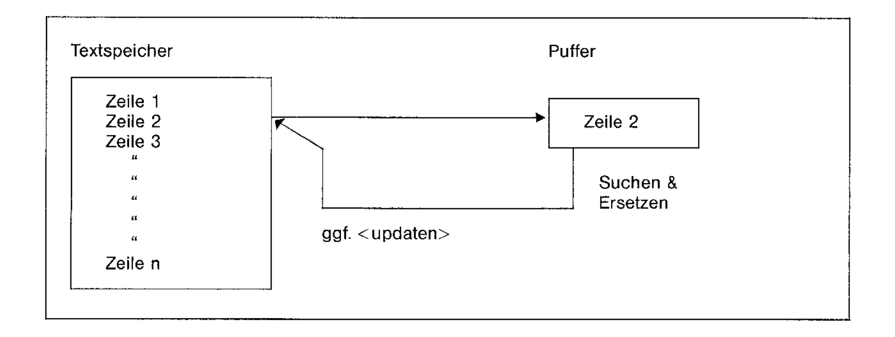


Die Verschiebung des Bildschirms wird über die Cursortasten gesteuert. In der Routine <crsrtast> im Abschnitt ‚Cursortasten' wird ein eingegebenes Zeichen auf den Code der Cursor-Tasten getestet. Ist eine der Tasten für Links- oder

Rechtsbewegung gedrückt, so wird der Cursor entsprechend weitergesetzt. Droht der Cursor außerhalb des Bildes zu geraten, so wird horizontal gescrollt, indem die <bild>-Routine aufgerufen wird. Bei Cursorbewegungen nach oben oder unten wird die Zeile verlassen und daher bei Bedarf <updaten> aufgerufen. Droht hier der Cursor das Bild zu verlassen, wo wird zunächst das Bild nach oben oder unten gescrollt. Dies erledigen Routinen im Betriebssystem des C 64. Sie werden unter den Namen <usrc> für das Scrollen nach oben, bzw. <dscr> für Scrollen nach unten aufgerufen. Damit die Routinen korrekt scrollen, müssen alle Zeilen des Bildschirms zunächst als Einfachzeilen (d.h. Zeilen mit weniger als 40 Zeichen) gekennzeichnet werden. Dies erledigt die Routine <einfachzei> im Abschnitt ‚Systemroutinen'. Um die Routine <uscr> benutzen zu können, muß außerdem eine Rücksprungadresse (hier ‚uscrrt') und der Bereich $AC–$AF, ‚scr-feld' genannt, auf den Stack gerettet werden. Die Routine <dscr> kann hingegen nach <einfachzei> aufgerufen werden. Dann wird am oberen bzw. unteren Rand die neue Zeile angezeigt. Das Programm merkt sich dabei die errechnete Adresse der Zeile in ‚adrud', so daß bei weiterem Scrollen in der Richtung die Adresse nicht jedesmal neu berechnet werden muß, sondern nur die Nachfolgezeile gesucht wird.

Die Abfrage der Funktionstasten erfolgt im Abschnitt ‚Funktionstasten' analog. Hier werden nur neue Zeilen berechnet und gegebenenfalls wieder <bild> aufgerufen.

Als letztes sind die Routinen zum Suchen und Ersetzen von Texten zu besprechen. Diese finden Sie im entsprechend gekennzeichneten Abschnitt. Es handelt sich dabei im Grunde nur um die Routine <replace> zum Ersetzen von Texten, wobei ihre Funktion bei Bedarf auf Suchen eingeschränkt wird. Dies geschieht bei einem Aufruf der Routine bei Label <find>. Als erstes versorgt sich die Routine mit Such- und Ersatzwort. Die Eingabe der Worte erfolgt in der Routine <argin>. Jede Zeile, in der gesucht werden soll, wird zunächst in den Puffer ‚edbuffer' übertragen. In dem Puffer wird dann das Suchwort gesucht und bei Bedarf durch das neue Wort ersetzt. Dabei darf die Gesamtlänge der Zeile die zulässigen 80 Zeichen nicht überschreiten. In diesem Fall wird nicht ersetzt. Kommt in dem Suchwort ein ‚?' vor, wird diese Stelle einfach nicht verglichen. Ist eine Zeile bis zum Ende durchsucht, wird sie, falls Ersetzen stattgefunden hat, mit Hilfe von <updaten> neu in den Text eingefügt. Auf jeden Fall wird in der nächsten Zeile weitergesucht.



4/5.4.4.2

Textfile-Aufbau

Um Änderungen am Assembler vornehmen zu können, muß man neben den wichtigsten Routinen auch die Daten kennen, mit denen sie arbeiten. Bei einem Assembler sind dies der Quelltext und die Verwaltung der Symbole. Wir wollen uns zunächst den Quelltext näher ansehen.

Der Text ist wesentlich einfacher als der Basix-Quelltext aufgebaut. Jede Zeile beginnt direkt mit dem Text der Zeile, der im Commodore-ASCII abgelegt wird. Das Ende der Zeile wird durch ein Nullbyte gekennzeichnet. Ist der Text beendet, folgen zwei weitere Nullbyte. Die Quellzeilen eines Assemblerprogramms werden dabei i.a. komprimiert (falls die Editoroption Pack eingeschaltet ist). Die Routine <pack> im Abschnitt ‚Text bearbeiten' entfernt dabei alle überflüssigen Leerzeichen. Im Gegensatz dazu fügt <unpack> die Leerzeichen wieder ein. Auf diese Weise lassen sich recht lange Quelltexte im Speicher unterbringen. Die Routinen <pack> und <unpack> arbeiten natürlich ebenfalls nur auf dem Puffer.

Beispiel:

```
Quelltext:   loop:   lda #1; Dies ist ein Test
                     txa
             end:    rts
```

```
Speicher:    loop:lda#1; Dies ist ein Test@txa@end:rts@@@
```

Wobei das Zeichen ‚@' hier für ein Nullbyte steht.

Auf Diskette steht der Text wie im Speicher. Jedoch wird er hier um zwei Byte ergänzt. In diesen Bytes wird die Dateilänge festgehalten. Dies erledigt die Routine <save> im Abschnitt ‚Save-Routine'. Beim Laden (Routine <load> im Abschnitt ‚Load-Routine') werden diese beiden Byte zuerst gelesen. So kann bereits vor dem Laden des Textes geprüft werden, ob dieser in den bereitgestellten Speicherplatz paßt.

Der Quelltext kann im gesamten Speicherplatz des C 64 liegen, d.h. auch im Bereich des Betriebssystems, Basic-Interpreters und der I/O-Bausteine. Um dies zu erreichen, wird während eines Zugriffes auf den Speicher der gesamte Speicher auf RAM umgeschaltet. Dies bringt jedoch Probleme mit sich, denn liegt das Programm als Modulversion vor, so wird auch das Modul abgeschaltet. Das Programm würde daher bei einem Speicherzugriff sofort abstürzen. Um dies zu umgehen, werden vom Programm Hilfsroutinen (siehe Programmabschnitt ‚RAM-Routinen') im RAM ab Abdresse ‚routinen' abgelegt. Dies sind die Routinen <ramcopy>, <holzg>, <putzg> etc., deren Adressen im Anschluß berechnet werden (siehe Listing). In diesen Routinen wird auf RAM umgeschaltet, der Speicherzugriff erledigt, wieder auf ROM geschaltet, und schließlich in das Programm zurückgesprungen. Zu diesen RAM-Routinen gehört jeweils ein kurzes Programm, das die Routine in das RAM überträgt (z.B. <putramro>, <putcopy> etc.).

4/5.4.4.3

Verwaltung des Symbolspeichers

Als nächstes betrachten wir den Aufbau des Symbolspeichers. Neben dem Symbolnamen soll der Wert und der Typ des Symbols in den Speicherbereich abgelegt werden. Für einen Eintrag in die Symboltabelle wurde daher folgender Aufbau gewählt:

<Symbolname> <Symboltyp> <Symbolwert>

mit: <Symbolname> ist die Bytefolge des Symbolnamens
<Symboltyp> ist ein Byte mit einem Wert von Null bis drei, wobei folgende Typenzuordnung gilt:

Bytewert	Typ
0	program, unused
1	fix, unused
2	program
3	fix

<Symbolwert> ist der Wert des Symbols, in zwei Byte gemäß der 6510 Reihenfolge, abgespeichert.

Ist also z.B. das Symbol ‚label' mit dem ‚.eq'-Befehl als $FFD2 definiert, und auch im Programm benutzt worden, so wird es mit folgender Bytefolge in die Tabelle eingetragen:

$4C $41 $42 $45 $4C	$03	$D2 $FF
Symbolname	Typ	Symbolwert

Das Ende der Symboltabelle wird mit einem Nullbyte gekennzeichnet.

Für das Eintragen von Symbolen ist die Routine <inslbl> im Abschnitt ‚Label Eintragen' zuständig. Sie trägt ein Symbol, welches unter der Adresse ‚buffer2' stehen muß, in die Symboltabelle ein. Als Wert wird dem Symbol der aktuelle Programmzähler PC zugewiesen. Gesucht werden die Symbole mit der Routine <sealbl> im Abschnitt ‚Labelwert suchen'. Diese Routine sucht ein Symbol, welches unter der Adresse ‚buffer2' steht, und liefert den Symbolwert in ‚labelwert' zurück. Aus Geschwindigkeitsgründen wird die eigentliche Suchroutine wieder in den RAM-Bereich ausgelagert (siehe auch Kapitel 4/5.4.4.2 Textfile-Aufbau). Als Beispiel wird angenommen, daß folgendes Programm assembliert wurde:

```
          .ba $c000
          .os
label 2:  .eq $1234
label 3:  .eq $ffd2

label1:   ldx #5
          lda #'0
          jsr label3

label4:   dex
          bne label1
          rts
```

Dann kann mit dem 'L'-Befehl im Hauptmenü folgende Symboltabelle ausgegeben werden:

```
label1    p c000    label2    u 1234
label3    f ffd2    label4    u c007
```

Die Ausgabe des Symbolspeichers erledigt dabei die Routine <listlbl> im Abschnitt ‚Label-File drucken'.

Diese Symboltabelle steht im Speicher normalerweise ab $E000 und kann mit dem eingebauten Monitor betrachtet werden:

```
:e000 4c 41  42 45 4c  31  02 00 label1..
:e008 c0|4c  41 42 45  4c  32 01 .label2.
:e010 34  12|4c 41 42  45  4c 33 4.label3
:e018 03  d2 ff |4c 41 42  45 4c .R.label
:e020 34  00 07 c0|00|ff  ff  ff 4.....
                   ↑
             Tabellenende-Byte
```

Zur Verdeutlichung der vier Einträge ist hier zwischen den Einträgen ein senkrechter Balken eingefügt.

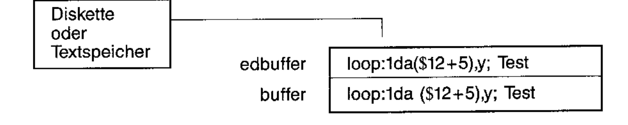
4/5.4.4.4

Übersetzen von Programmzeilen

Im Abschnitt ‚Assemblieren' beginnt nun der eigentliche Assembler. Er wird über die Routine <assemble> für einen vollständigen Assemblerlauf, oder über <pass2> gestartet, falls nur der zweite Paß gewünscht wird. Beim Assemblieren werden zunächst mit Hilfe der Routine <passpara> die benötigten Parameter eingelesen, also Dateinamen, Protokoll drucken etc. Die Routine <passbegin> öffnet dann die benötigen Dateien, und setzt Speicherstellen wie Programmzähler u.ä. auf ihre Anfangswerte. Für jeden Paß wird dann einmal die Routine <pass> im Abschnitt ‚Einen Paß ausführen' aufgerufen. Nach der Ausgabe von Assemblierzeit, Fehleranzahl und Endadresse ist die Aufgabe von <assemble> dann erledigt.

Für die Assemblierung des Textes sorgt die Routine <pass>. In ihr wird der Text zeilenweise aus dem Textspeicher oder von Diskette in die Puffer ‚edbuffer' und ‚buffer' eingelesen. Die eingelesene Zeile wird nun von der Routine <syntax> im Abschnitt ‚Syntaxanalyse' aufgeteilt. Ein evtl. vorhandenes Symbol wird im Puffer ‚buffer2' abgespeichert und die Adressierungsart im Puffer ‚adrmotxt'. Der Rest der Zeile bleibt im Puffer ‚buffer'. Der Text in ‚edbuffer' wird nicht verändert. Er wird zur Ausgabe bei Protokollausgabe benutzt. Die folgende Graphik soll diese Aufteilung verdeutlichen:

1. Einlesen einer Textzeile



2. Aufteilung durch <syntax>

edbuffer:	loop:1da($12+5),y; Test
buffer:	1da$12+5
buffer2:	loop
adrmotxt:	(),y

Im ersten Paß wird das Symbol in die Symboltabelle eingetragen, im zweiten Paß hingegen ignoriert. Als nächstes müssen nun Befehl, Adressierungsart und Operand ausgewertet werden. Der Wert des Operanden wird eigentlich erst im zweiten Paß benötigt, da der Assembler aber die Zero-Page Adressierung automatisch erkennen soll, muß der Operand bereits im ersten Paß ausgewertet werden.

Diese Auswertung übernimmt die Routine <ausdruck> im Abschnitt ‚Operand berechnen'. Sie wertet arithmetische Ausdrücke mit den Grundrechenarten ‚+', ‚-', ‚*' und ‚/', die ab ‚buffer+3' stehen von links nach rechts aus. Ist der Ausdruck noch undefiniert, wird in der Speicherstelle ‚labeltyp' Bit 7 gesetzt. Andernfalls steht der Wert des Ausdrucks in ‚zahl1'.

Die Adressierungsart wird durch die Routine <getadrmode> im Abschnitt ‚Adressierungsart' aus dem Text unter ‚adrmotxt', und dem bereits berechneten Operanden gewonnen. Sie legt unter ‚adrmode' die Adressierungsart wie folgt codiert ab:

Code	Adressierungsart
1	implizit oder Akkumulator
2	Unmittelbar
3	Relativ
4	Indirekt
5	(Indirekt,X)
6	(Indirekt),Y
7	Absolut
8	Zero-Page
9	Absolut,X
10	Zero-Page,X
11	Absolut,Y
12	Zero-Page,Y

Die Adressierungsart Relativ (Nr. 3) wird nicht erkannt, sondern als Absolut (Nr. 7) zurückgegeben.

Als nächstes wird der Befehlscode ermittelt. Die Routine <getcode> im Abschnitt ‚Code ermitteln' sucht in der Tabelle der 6510-Befehle den Befehlscode. Wurde Zero-Page Adressierung erkannt, aber diese ist bei dem angegebenen Befehl nicht möglich, wird auf die längere Adressierung umgeschaltet, was durch einfaches Erniedrigen der Adressierungsarten 8, 10 und 12 um eins erreicht wird. Die Adressierungsart Relativ wird in <getcode> ebenfalls korrigiert. Der ermittelte Befehlscode wird in ‚order' gespeichert.

Damit ist die Zeile komplett übersetzt. Nun muß der erzeugte Code noch in den Speicher und/oder auf Diskette gebracht, und ggf. ein Protokoll gedruckt werden. Diese Arbeit erledigt die Routine <codeout> im Abschnitt ‚Code ausgeben'.

Bei einem Fehler bei der Übersetzung wird die Routine <asserror> aufgerufen. Sie gibt den Fehler auf, erhöht den Fehlerzähler um eins, und wartet ggf. auf ein Kommando von der Tastatur (bei .wa).

Der bisher gezeigte Übersetzungsvorgang gilt natürlich nur für die offiziellen 6510 Mnemonics. Die von dem Assembler ebenfalls beherrschten Pseudo-Befehle müssen anders behandelt werden. Bei einem Pseudo-Befehl wird zunächst nur ein evtl. vorhandenes Symbol in die Symboltabelle eingetragen. Dann wird die weitere Arbeit der Routine <expse> im Abschnitt ‚Pseudo-Befehle' überlassen. Diese sucht zunächst den Befehl in der Tabelle ab ‚pscol'. Dann wird die Adresse der zugehörigen Routine aus einer Tabelle geholt. Da manche Pseudo-Befehle im ersten Paß anders abgehandelt werden, als im zweiten Paß, wird die Adresse der zum Befehl gehörenden Routine im ersten Paß aus der Tabelle ab ‚jupa1', und im zweiten Paß aus der Tabelle ab ‚jupa2' geholt. Die Ausführung dieser Routinen schließt dann die Übersetzung ab.

4/5.4.4.5

Verändern des Assemblers

Mit den angegebenen Hinweisen auf die Arbeitsweise des Assemblers sollte es Ihnen möglich sein, den Assembler selbst zu verändern, indem Sie zum Beispiel weitere Pseudo-Befehle einbauen oder erweiterte Editiermöglichkeiten. Im einfachsten Fall kann aber auch nur eine Veränderung der Standardparameter nötig sein.

Wie der Assembler eingegeben werden kann, wurde bereits im Kapitel 4/5.4.3.1 gezeigt. Wenn Sie aber bereits im Besitz des Assemblers sind (Diskette vom Verlag), und diesen verändern wollen, sollten Sie wie folgt vorgehen:

1. Kopieren der Quelltexte und ASS.RAM auf eine andere Diskette, damit Sie weiterhin über das Original verfügen. Dies ist aber insbesondere auch bei der vom Verlag gelieferten Diskette nötig, da diese schreibgeschützt ist.
2. Starten von ASS.RAM
3. Mit dem ‚O'-Befehl den Symbolspeicher auf 9000-FFFF setzen, da der Standardsymbolspeicher für die Anzahl der verwendeten Symbole nicht ausreicht.
4. mit ‚A' assemblieren starten, als Dateinamen ASS.SRC angeben.
5. Die Abfragen beantworten. Der Assembler erstellt daraufhin eine neue Version mit dem Namen ASS.EXE auf der Diskette.
6. ggf. der Diskette einen Validate-Befehl senden, um die bei der Floppy 1541 immer wieder vorkommenden falschen Blockbelegungen zu korrigieren.
7. ggf. den Symbolspeicher wieder auf normale Werte setzen.
8. ggf. ASS.RAM löschen und ASS.EXE in ASS.RAM umbenennen.

4/5.5

Disassembler

Als erste Erweiterung unseres Assemblersystems wollen wir einen Disassembler vorstellen, der sich leicht in das System einbinden läßt.

Das Programmm befindet sich auf der Grundwerksdiskette

4/5.5.1

Aufgabe des Disassemblers

Der Disassembler hat die Aufgabe ein fertiges Objekt-Programm wieder in einen lesbaren Text zu verwandeln. Der hier vorgestellte Disassembler wird vollständig in das System eingebunden, und kann über den Menüpunkt ‚Monitor' vom Hauptmenü aus gestartet werden. Er erwartet dann die Eingabe einer Startadresse als vierstellige Hexadezimalzahl. Daraufhin beginnt er den Speicherinhalt ab der eingegebenen Adresse als Klartext auszugeben. Er kann dann mit der <SHIFT>-Taste angehalten werden. Die Ausgabe wird mit der <C= >-Taste fortgesetzt. Mit der <STOP>-Taste wird der Disassembler verlassen.

Natürlich ist die Ausgabe des Disassemblers nicht mit dem Quelltext eines Assemblerprogramms zu vergleichen. Zum einen liefert der Disassembler die physikalischen Adressen von Sprungzielen und Daten, und keine symbolischen Adressen, und zum anderen kann der Disassembler nicht zwischen Programm und Daten unterscheiden. Läuft er also beim disassemblieren in einen Datenbereich hinein (z.B. eine Zeichenkette), so versucht er diese Daten zu übersetzen. Gelingt ihm das, zeigt er Befehle an, die allerdings von dem Programm niemals ausgeführt würden, andernfalls gibt er drei Fragezeichen aus.

Als Beispiel soll das folgende Programm dienen:

```
        .os
        .wa
        .ba $c000
start:  lda #'3
        sta help
        dex
        bne start
        rts
        .by "dies ist text"
help:   .by 0
```

Wenn man dieses Programm übersetzt, steht es ab der Adresse $C000 im Speicher. Nun kann man mit ‚N' vom Hauptmenü aus den Disassembler starten. Als nächstes wird durch Eingabe von „C000" die Startadresse festgelegt. Der Disassembler liefert daraufhin die folgende Ausgabe:

```
C000 A9 33       LDA  #$33      Disassemblierter Programmbereich
C002 8D 16 C0    STA  $C016
C005 CA          DEX
C006 D0 F8       BNE  $C000
C008 60          RTS
C009 44          ???            Übersetzung des Datenbereichs
C00A 49 45       EOR  #$45      führt zu falschen Befehlen
C00C 53          ???            und Fehlermeldungen ('???')
C00D 20 49 53    JSR  $5349
C010 54          ???
C011 20 54 45    JSR  $4554
C014 58          CLI
C015 54          ???
C016 00          BRK
```

4/5.5.2

Arbeitsweise des Disassemblers

Der Disassembler arbeitet der Einfachheit halber mit der Befehlstabelle des Assemblers, in der ja alle Befehle mit ihren Adressierungsarten in Klartext und verschlüsselt eingetragen sind.

Zunächst wird jedoch die Startadresse des zu assemblierenden Speicherbereichs übernommen. Dies erledigt die Routine <hexbereich>. Anschließend beginnt ab ‚disloop‘ die Hauptschleife des Programms, welche als erstes die Adresse des gerade assemblierten Bereichs ausgibt. Der unter dieser Adresse stehende Code wird in der Speicherstelle ‚buf‘ abgelegt, und als zweistellige Hexadezimalzahl durch die Routine <ausgeben> ausgegeben. Anschließend wird der Code in der Befehlstabelle gesucht. Dies geschieht ab ‚seaopcodel‘. War die Suche erfolgreich, findet man ab ‚buf'+1 den Befehlsnamen, und in ‚buf'+4 die Nummer der Adressierungsart. Andernfalls werden drei Fragezeichen ausgegeben, und die Disassemblierung bei der nächsten Adresse fortgesetzt.

Bei erfolgreicher Suche kann nun die Anzahl der zum Befehl gehörenden Byte aus der Tabelle ‚arganz' entnommen werden.

Gehören noch einige Byte zum Befehl, werden diese zunächst ausgegeben, und ab ‚buf‘+6 gespeichert. Bei einem Sprungbefehl wird statt der Sprungweite die Zieladresse berechnet (ab ‚goname‘) und in ‚buf‘+6 sowie ‚buf‘+7 gespeichert.

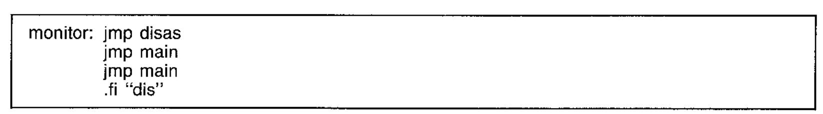
Nun kann der Befehlsname ausgegeben werden. Um auch die Adressierungsart korrekt auszugeben, sind ab ‚adrtxt‘ die Texte der einzelnen Adressierungsarten abgelegt, jedoch ohne irgendwelche Werte. Für die Adressierungsart Zero-Page,Y lautet der Text also ‚($),Y‘. Dieser Text wird nun ausgegeben. Dabei wird überprüft, ob das Dollarzeichen ausgegeben wurde. Ist dies der Fall, muß an dieser Stelle der Operand, welcher ab ‚buf‘+6 abgespeichert wurde, ausgegeben werden. Ab ‚patend‘ wird nun die nächste zu disassemblierende Adresse berechnet. Die Routine <moabort?> testet abschließend, ob der Vorgang gestoppt oder abgebrochen werden soll. Ist dies nicht der Fall, kann der Vorgang bei der Hauptschleife fortgesetzt werden.

4/5.5.3

Einbinden in das Assembler-System

Der Disassembler wird auf einfache Weise in das Assemblersystem eingebunden. Im Quelltext ‚PSEUDO‘ existieren drei Sprungbefehle, die für eine Monitorerweiterung vorgesehen sind. Einen dieser Sprungbefehle werden wir nutzen, um über den Monitor-Befehl im Hauptmenü in den Disassembler zu springen. Dies ist jedoch nur eine Übergangslösung. Wenn wir später die Monitorerweiterung veröffentlichen, wird der Disassemblerbefehl in den Monitor mit aufgenommen.

Die Erweiterung geschieht bei dem Basic-Lader durch einfaches Hinzufügen von Programmzeilen. Dieses Verfahren ist im Kapitel 4/5.7 beschrieben. Wer mit dem Quelltext des Disassemblers arbeitet, muß zunächst den Quelltext des Disassemblers eingeben, und unter den Dateinamen ‚DIS‘ auf Diskette speichern. Anschließend wird die Quelltextdatei ‚PSEUDO‘ geladen, um die Sprungvektoren zu ändern. Mit der Funktionstaste ‚F7‘ gelangen Sie an das Textende. Dort finden Sie hinter dem Symbol ‚monitor‘ die drei Sprungbefehle. Diese müssen auf die Disassemblererweiterung eingestellt werden. Außerdem muß die Datei mit dem Ladebefehl für die Quelltextdatei ‚DIS‘ angeschlossen werden. Die Sequenz hinter ‚monitor‘ muß also wie folgt aussehen:



```
monitor: jmp disas
         jmp main
         jmp main
         .fi "dis"
```

Wer die Diskette des Verlags bestellt hat, braucht natürlich nichts zu ändern. Hier sind die Disassembler- und die Monitorerweiterung bereits eingebunden.

4/5.6

Monitor

Als zweite Erweiterung wollen wir einen kleinen Monitor vorstellen, der es ermöglichen soll, weitergehende Operationen im Speicher des C 64 vorzunehmen. Auch er soll wieder vollständig in das bestehende System eingebunden werden, so daß ein Aufruf vom Hauptmenü aus möglich ist. Den bisher unter dem Menüpunkt „Monitor" eingebundenen Disassembler werden wir als Befehl in den Monitor mit aufnehmen.

Damit wird unser Assemblersystem zu einem kompletten kleinen Entwicklungssystem für 6502-Maschinenprogramme auf dem C 64. Wer die Möglichkeit hat, sollte das Programm als Modulversion betreiben. Nur so kommt die volle Leistungsfähigkeit, der Zugriff auf alle Speicherstellen in RAM, ROM und I/O zum Tragen. Insbesondere die Möglichkeit, den Assembler auf einer mit einer Batterie gepufferten RAM-Modulplatine zu betreiben, macht die Arbeit mit dem Assembler sehr bequem.

Beachten Sie bei der Modulversion bitte, daß ein 8-KByte-Modul nicht ausreicht, um den Assembler aufzunehmen. Im 16-KByte-Modul ist hingegen noch genügend Platz für eigene Erweiterungen. Das Erstellen der Modulversion ist übrigens nur mit Hilfe des Quelltextes möglich. Der Basic-Lader erzeugt immer die RAM-Version.

Das Programmm befindet sich auf der Grundwerksdiskette

4/5.6.1

Aufgabe des Monitors

Ein Monitor dient zur Fehlersuche in Programmen. Er stellt für diese Aufgabe eine Reihe von Befehlen zur Verfügung, die der Speichermanipulation dienen. Mit Hilfe seiner Befehle ist es möglich, jedes Byte im Speicher des C 64 zu verändern, Pro-

gramme aufzurufen und sich die Registerinhalte der CPU anzusehen. Unser Monitor erlaubt darüber hinaus, auch die Informationen auf der Diskette zu verändern.

4/5.6.2

Grundroutinen zur Speicherveränderung

Unser Monitor enthält mehrere Routinen zur Speicherveränderung. Die einfachste dieser Routinen ist die Blockverschieberoutine, welche mit dem „V"-Befehl aufgerufen wird. Diese Routine ist praktisch schon im Editor enthalten, um die Textteile beim Einfügen und Löschen von Zeilen zu verschieben. Wir brauchen der Editorroutine <move> also nur noch den zu verschiebenden Bereich übergeben. Dies wird in der Routine <verschieben> des Monitors durchgeführt. Sie liest zunächst die drei Eingabewerte ein und ruft dann <move> auf.

Die anderen Routinen zur Speicherveränderung dienen dem Ändern einzelner Bytes im Speicher. Am häufigsten wird von diesen Befehlen der „M"-Befehl benötigt. Er zeigt einen Speicherbereich auf dem Bildschirm an. Ein Beispiel:

```
:A000   94  E3  7B  E3  43  42  4D  42   .c{cCBMB
:A008   41  53  49  43  30  AB  41  A7   ASICO(A'
:A010   1D  AD  F7  AB  A4  AB  BE  AB   .-w($+>+
:A018   B0  B0  05  AC  A4  A9  9F  AB   .0.,$).(
```

Diese Ausgabe wird von der Routine <memor> erzeugt. Sie gibt zunächst am Anfang der Zeile einen Doppelpunkt aus, auf den wir später noch zu sprechen kommen. Dann wird die momentan gültige Adresse, welche sich unter der Adresse

„zg" findet, ausgegeben. Nun werden in der Schleife „memol" acht folgende Byte aus dem Speicher geholt, als zweistellige Hexadezimalzahl ausgegeben und ab „buf" gespeichert. Am Ende der Zeile werden nun noch die acht im „buf" gespeicherten Werte als Zeichen ausgegeben. Dies wird von der Routine <zeichnen> erledigt. Diese Ausgabe wird wiederholt, bis die Stoptaste gedrückt oder die Endadresse erreicht wurde. Damit ist die Aufgabe des „M"-Befehls praktisch beendet. Um die angezeigten Werte auch ändern zu können, steht am Zeilenanfang der Doppelpunkt. Wird nämlich ein Wert in einer Zeile geändert und die <RETURN>-Taste gedrückt, so erscheint dies dem Computer als Eingabe einer Zeile. Im Beispiel sei dies die letzte der Zeilen:

```
:A018   81   B0   05   AC   A4   A9   9F   A8   .0.,$).(
```

Der Doppelpunkt dient daher als Befehl, genau wie das „M" des anzeigenden Befehls. Wird der Doppelpunkt vom Computer gelesen, so springt er in die Routine <doppelpunkt>. In dieser wird zunächst die Adresse der Bytes eingelesen werden und dann die acht folgenden Werte. Diese werden dann ab der angegebenen Adresse gespeichert.

Zwei weitere Routinen, die eine Speicherveränderung ermöglichen, arbeiten nach dem gleichen Prinzip. Nur geben sie keine Hexadezimalzahlen aus, sondern die Werte als ASCII-Zeichen („A"-Befehl), oder als POKE-Code („C"-Befehl). Die Ausgaben werden von den Routinen <asciitext> bzw. <codetext> erzeugt. Im Beispiel für die Ausgaben dieser Routinen können Sie erkennen, daß dem „A"-Befehl ein Komma vorangestellt ist, während dem „C"-Befehl das Zeichen „ ' " vorangestellt wird.

```
A-Befehl:   1332  ."......P.L....W...X.''...)...,
            ,1352 ..%..OH. .%...%.....P.J-X.H-W.H
            ,1372 .".&.&..)..O '.(JP.&.&..%.H%.H".
            ,1392 %.)...OH. 8. 8. 8.%...P.. '.JP..
            ,13B2 ......F....).....  ..I ..I$.>I%.&
```

```
C-Befehl:   '1332  ....U....P.LR..MW..MX...F...E. .
            '1352  .....OH.  ....E...E...P.J.X.H.W.H
            '1372  ...&.&.....O  '.(JP.&.&....H..H..
            '1392  ..)...OH.  ..  ..  ......P..  '.JP..
            '13B2  E..E..F......E.E.  ..I  ..I$..I%.&
```

Wird eine der Zeilen verändert, werden die neuen Werte von der Routine <mokomma> (bzw. <tick>) wieder eingelesen. Dabei werden die Punkte als nicht eindeutig anzuzeigende Zeichen überlesen. Gespeichert werden daher nur die eindeutigen Zeichen.

4/5.6.3

Aufruf von Programmen

Das Aufrufen von Programmen kann auf zwei Arten erfolgen: Für Programme, welche mit dem RTS-Befehl abgeschlossen wurden, dient der „U"-Befehl. Das Programm wird dann über einen JSR-Befehl aufgerufen. Als zweite Möglichkeit kann ein Programm über den „G"-Befehl aufgerufen werden (mit Hilfe eines JMP-Befehls). Um in den Monitor zurückkehren zu können, muß das Programm dann einen BRK-Befehl enthalten.

Der „G"-Befehl wird von der Routine <go> abgearbeitet. Zu ihren Aufgaben gehört es, den Sprungbefehl „JMP Adresse" in die RAM-Routine zu kopieren. Dann werden die Register der CPU mit den Startwerten geladen, welche mit dem „R"-Befehl festgelegt werden können (siehe 4/5.6.5 „Weitere nützliche Routinen") Über die RAM-Routinen wird dann die gewünschte Adresse angesprungen.

Stößt der Computer bei der Abarbeitung der durch „G" aufgerufenen Routine auf den BRK-Befehl, so springt er zu der Adresse „break" in den Monitor. Hier werden dann die aktuellen Registerinhalte der CPU in „buffer3" gespeichert und die Register mit Hilfe der Routine <register> ausgegeben.

Der „U"-Befehl wird hingegen von der Routine <unterprg> abgewickelt. Er funktioniert analog zum „G"-Befehl. Allerdings wird hier ein „JSR Adresse" in die RAM-Routine kopiert und dann die Routine aufgerufen. Nach einem RTS-Befehl wird so der Monitor bei der Adresse „monirt" fortgesetzt. Hier werden wieder die Register in „buffer3" gerettet und mit Hilfe der Routine <register> ausgegeben.

4/5.6.4

Suchen mit dem Monitor

Ein recht nützlicher Befehl ist der Suchbefehl „F", welcher das Suchen von Bytefolgen im Speicher des C 64 ermöglicht. Dieses Suchen von Bytefolgen wird von der Routine <mofind> erledigt, welche sich zunächst die Adressen des Speicherbereichs holt, in dem gesucht werden soll. Alle weiteren Eingaben werden als zu suchende Bytefolge interpetiert. Diese wird ab „mofiarg" in „buffer" eingelesen. Jede in Hochkomma eingeschlossene Zeichenfolge wird dabei von <ttxtarg> in den Buffer übertragen. Alles andere wird als Folge von zweistelligen Hexadezimalzahlen aufgefaßt, und als solche eingelesen.

Das Suchen der Bytefolge geschieht dann ab „seastr", indem die Bytefolge in „buffer" mit der im zu untersuchenden Speicherbereich stehenden Bytefolge byteweise verglichen wird. Stimmen sie überein, wird die Anfangsadresse des Bereichs ausgegeben (ab „mofound") und weitergesucht. Sonst wird die Anfangsadresse des Bereichs um eins erhöht und weitergesucht bis die Endadresse erreicht ist. So bekommen Sie eine Liste aller Anfangsadressen der gesuchten Bytefolge im angegebenen Bereich.

4/5.6.5

Weitere nützliche Routinen

Als weiteren wichtigen Befehl betrachten wir den „R"-Befehl, welcher von der Routine <register> repräsentiert wird. Er wurde gelegentlich schon angesprochen. Der „R"-Befehl gibt die Register der CPU in der folgenden Form aus:

```
PC     AC  XR  YR  SP  NV-BDIZC  CHL  SR  PP
;0000  08  00  0C  F6  00110000  111  30  37
```

Die Bedeutung der Ausgaben wurde bereits im Kapitel 4/5.4.2.10 beschrieben.

Durch das Voranstellen des Semikolons können diese Werte auch verändert, und wieder eingelesen werden. Auf diese Weise lassen sich den Registern Startwerte für die mit „G" oder „U" aufgerufenen Routinen übergeben.

Durch Ändern der Werte für die internen Leitungen Char/Hiram und Lowram wird außerdem festgelegt, auf welche Speicherbereiche sich die Routinen des Monitors beziehen. Auf einfachste Weise kann man sich so ROM und RAM des C 64 ansehen.

Das Einlesen der veränderten Daten erledigt die Routine <regsin>. Sie holt sich die angegebenen Daten und speichert sie neu ab. Anschließend werden die Daten über die Routine <register> neu angezeigt.

Der Befehl „=" dient dem Vergleichen von Speicherbereichen und startet die Routine <ramvergl>. Die beiden Adressen werden eingelesen und in „zahl1" bzw. „zahl2" gespeichert. In der Vergleichsschleife ab „doramver" werden dann die Daten unter den angegebenen Adressen geholt und miteinander verglichen. Bei gleichen Daten werden die Adressen erhöht und die Schleife fortgesetzt. Sonst werden die aktuellen Adressen ausgegeben (ab „ungleich").

Mit dem „P"-Befehl ist es möglich, den Drucker die Arbeit mit dem Monitor protokollieren zu lassen. Die Routine <priplumi>, welche den „P"-Befehl abarbeitet, erwartet als Eingabe ein „+" bzw. „-" für Ein- bzw. Ausschalten des Druckerprotokolls. Zum Einschalten wird ab „priplu" der Druckerkanal eröffnet und die Flagge für „Ausgabe auf Drucker" gesetzt. Beim Ausschalten wird der Kanal wieder geschlossen und die Flagge zurückgesetzt.

Der „D"-Befehl ersetzt die vorläufige Einbindung des Disassemblers in das Assemblersystem. Der Disassembler wird also in Zukunft mit dem „D"-Befehl vom Monitor aus aufgerufen.

Mit dem „X"-Befehl gelangen Sie schließlich wieder über die Routine <exit> in das Hauptmenü.

4/5.6.6

Diskettenroutinen

Zum Ein- und Auslesen von Speicherbereichen auf bzw. von Diskette dienen der „L"- bzw. „S"-Befehl. Mit diesen Befehlen können Bildschirminhalte oder Programme gelesen und gespeichert werden. Auch das Laden von Programmen an eine andere als vom Programm vorgegebene Startadresse ist hiermit möglich.

Der „L"-Befehl startet die Routine <monload>. Diese testet zunächst, ob eine Adresse folgt. Ist dies der Fall, wird sie eingelesen und als Startadresse benutzt. Andernfalls wird das Programm bei „absload" fortgesetzt. Nach dem Einlesen des Dateinamens kann dann die Datei eröffnet werden. Aus dieser wird zunächst die Startadresse gelesen. Hat der Benutzer beim Aufruf bereits eine Adresse angegeben, wird die aus der Datei gelesene Adresse nicht weiter beachtet, sonst ist diese die gültige Startadresse. Dann wird in der Schleife ab „lorel" die Datei eingelesen. Zum Schluß muß die Datei nur noch geschlossen werden. Als zusätzliche Information für den Benutzer wird noch die Start- und Endadresse des gelesenen Bereichs ausgegeben.

Die Routine <monsave> repräsentiert den „S"-Befehl. Sie ließt zunächst den Dateinamen ein. Anschließend verlangt sie die Start- und Endadresse des zu speichernden Bereichs. Nach dem Eröffnen der Datei wird die Startadresse dann in die Datei übertragen. Ab „mosa" wird nun noch der Speicherbereich in die Datei geschrieben.

Neben den gewöhnlichen Befehlen bietet unser Monitor noch einige Befehle zum Manipulieren von Daten auf der Diskette. Wie Sie sicherlich wissen, sind die Daten auf der Diskette in Blöcke zu je 256 Byte eingeteilt. Diese Blöcke verteilen sich auf 35 Spuren (Tracks), die je eine unterschiedliche Anzahl von Blöcken (dann Sektoren genannt) aufnehmen. Die Diskettenroutinen des Monitors basieren auf der Idee, jeweils einen Block in einen Buffer ab $CF00 einzulesen, zu verändern und wieder auf Diskette schreiben zu können. Als kleine Hilfe zeigt die folgende Tabelle kurz die Organisation der Floppy 1541 an:

Track		Sektoren	
dezimal	hex	dezimal	hex.
1..17	01..11	0..20	00..14
18..24	12..18	0..18	00..12
25..30	19..1E	0..17	00..11
31..35	1F..23	0..16	00..10

Der Track 18 ist für das Inhaltsverzeichnis der Diskette reserviert. In Track 18 Sektor 0 werden die belegten Blöcke markiert, und ab Sektor 1 folgen die Namen und die Startblöcke der Dateien auf der Diskette.

Zum Lesen eines der Blöcke wird der „B"-Befehl benutzt. Dieser Befehl wird von der Routine <readbuffer> ausgeführt. Sie eröffnet zunächst den Befehlskanal der Diskettenstation über der Routine <kanal 15>. Dann wird mit Hilfe der Routine <sector> der gewünschte Track und Sektor eingelesen. Stellt <sector> fest, daß keine Eingabe gemacht wurde, wird von der Routine <logicsec> der logisch nächste Sektor ermittelt. Über den Blocklesebefehl der Diskette wird dann der Block in den Puffer eingelesen.

Analog wird beim Schreiben des Blocks auf Diskette vorgegangen. Der „W"-Befehl startet die Routine <writebuffer>. Hier wird zunächst getestet, ob eine weitere Eingabe folgt. Ist dies nicht der Fall, wird die Routine bei „sameblock" fortgesetzt. Sonst wird der gewünschte Track und Sektor von der Routine <sector> eingelesen. Dann wird über den Blockschreibbefehl der Block auf Diskette zurückgeschrieben.

Um die im Puffer stehenden Daten manipulieren zu können, könnte man über den „M"-Befehl den Buffer ansprechen. Dies geht jedoch kürzer mit dem „E"-Befehl, der eine Abkürzung des „M"-Befehls darstellt, und daher auch über die gleichen Routinen abgearbeitet wird. Er verlangt für die Adressen nur noch das Lowbyte der vollständigen Adresse und gibt diese bei der Ausgabe auch nur aus. Ein Beispiel für die Ausgabe des „E"-Befehls sehen Sie im folgenden Bild:

```
.00  12  04  83  14  01  2A  2A  54    ......**T
.08  41  4E  4B  20  41  54  54  41    ANK ATTA
.10  43  4B  2A  2A  2A  00  00  00    CK***...
.18  00  00  00  00  00  00  01  00    ........
.20  00  00  83  14  02  3E  20  20    .....>
.28  51  55  45  4C  4C  54  45  58    QUELLTEX
.30  54  20  20  20  3C  00  00  00    T<     ...
.38  00  00  00  00  00  00  01  00    ........
.40  00  00  82  11  00  54  41  4E    .....TAN
```

Die Daten können wieder verändert werden, und werden dann an die richtige Adresse im Puffer geschrieben. Der Monitor erkennt die verkürzte Eingabe an dem Punkt am Anfang der Zeile.

Eventuell bei der Arbeit mit der Diskettenstation auftretende Fehlermeldungen können Sie sich mit dem „@"-Befehl ansehen. Die Routine <flomeld> gibt daraufhin den Fehlertext aus. Sie benutzt dazu die im Assemblersystem bereits vorhandene Routine <diskerr>.

Als kleine Hilfe existiert noch der „I"-Befehl, welcher über die Routine <info> den gerade eingestellten Track und Sektor als zwei zweistellige Haxadezimalzahlen ausgibt.

4/5.6.7

Einbinden in das Assemblersystem

Das Einbinden in das Assemblersystem geschieht wieder auf die gleiche Weise, wie bei der Disassemblererweiterung. Die drei Sprungbefehle am Ende der Quelldatei „PSEUDO" werden auf die Monitorerweiterung eingestellt. Sie müssen daher folgendermaßen abgeändert werden:

```
monitor:    jmp  monstart
            jmp  monirt
            jmp  break
            .fi  „mon"
```

Der Pseudobefehl „.fi „mon"" sorgt für das Anhängen des Monitorquelltextes. Der Quelltext des Disassemblers wird anschließend vom Quelltext „MON" aus eingeladen, so daß der Monitor praktisch zwischen dem ursprünglichen Assemblersystem und dem Disassembler eingeschoben wird.

4/6

Utilities

Trotz mannigfacher Problemlösungen aus den unterschiedlichsten Bereichen gibt es immer wieder Programmteile, die in den verschiedensten Programmen eingesetzt werden können. Im allgemeinen hat sich für solche Programmsequenzen der englische Begriff Utilities eingebürgert. Innerhalb von Kapitel 4/6 möchten wir für Sie nach und nach eine Unterprogrammsammlung aufbauen, mit der Sie dann Teilbereiche Ihrer Probleme lösen können.

Zunächst wollen wir dabei auf allgemeine Algorithmen eingehen, wie z.B. Sortierverfahren, das Selektieren von Daten, Menüstrukturen oder die Verarbeitung von Kalenderdaten. Dies sind sicherlich die gebräuchlichsten Einsatzbereiche, die zudem noch rechnerunabhängig sind. Als nächstes folgen zwei Kapitel, wo wir Utilities aus den verschiedensten Bereichen aufgrund der Programmierweise zusammengefaßt haben: Basic-Utilities und Assembler-Utilities.

Da der C 64 im Verhältnis zu seinen Eigenschaften über ein sehr mageres Basic verfügt, wollen wir auch Basic-Erweiterungen im Rahmen der Utilities vorstellen. Zunächst in Kapitel 4/6.4 für den C 64, in Kapitel 4/6.5 aber auch für den C 128, da trotz des umfangreicheren Basic des C 128 sicherlich noch einiges wünschenswert ist. Im Rahmen dieser beiden Kapitel werden wir auch die Vorgehensweise zur Erweiterung der Basic-Varianten besprechen.

Utilities finden Sie jedoch nicht nur in Kapitel 4/6, sondern auch innerhalb der Musterprogramme in Teil 5, sowie hauptsächlich im Rest von Teil 4, bei den Beispielen zu den verschiedenen Kursen.

4/6.1
Basic-Utilities

Nachdem wir Kapitel 4/6.2 Algorithmen von allgemeiner Bedeutung gewidmet haben, möchten wir im vorliegenden Teil speziell auf Basic-Utilities eingehen. Die Basic-Utilities wollen wir auch wieder nach Themengebieten unterteilen, zunächst nach den Gebieten Grafik, Sound und mathematische Utilities, weitere werden folgen.

4/6.1.1
Grafik-Utilities

Durch die Grafikfähigkeiten, die die Home-Computer heutzutage allgemein haben, ist es auch dem Heimanwender erlaubt, künstlerische oder statistische Grafiken zu erzeugen. Dabei tauchen des öfteren die gleichen Probleme auf. Mit den folgenden Routinen wollen wir Ihnen die Möglichkeit in die Hand geben, Ihre Probleme effizient zu lösen.

Da Sprites und Shapes nur besondere Darstellungsformen von Grafiken sind, haben wir diese ebenfalls im vorliegenden Kapitel zusammengefaßt.

4/6.1.1.1

Sprites spiegeln

Manchmal ist es notwendig, Sprites zu entwerfen, die bis auf eine spiegelbildliche Darstellung vollkommen identisch sind. In diesem Buch verwenden wir zum Beispiel gespiegelte Sprites in unserem Spritekurs für den C 128 in Kapitel 4/4.3.2. Deshalb wollen wir im Folgenden zwei kurze Programme vorstellen, die ein Sprite einerseits an der X-Achse, und andererseits an der Y-Achse spiegeln, was sicherlich nicht nur für unsere Anwendung interessant ist.

Die Beispiele sind für den C 128 gedacht, da hier die Spritedaten an eine Zeichenreihe übergeben werden können. Prinzipiell ist eine Spiegelung nach dem gleichen Schema auch im 64er-Modus möglich. Die Beispiele sind ebenfalls dem Spritekurs des C 128 entnommen.

Zunächst wollen wir uns die Spiegelung an der X-Achse vornehmen, da dies die einfachere Vorgehensweise ist. In diesem Fall muß die erste Zeile zur letzten Zeile werden und die letzte Zeile zur ersten, wenn man sich ein Sprite in Zeilen aufgelöst darstellt. Dies kann man sehr einfach mit der MID$()-Anweisung vollziehen, wenn man – die letzten vier Zeichen (= Byte) einmal außer Acht lassend – jeweils von hinten drei Zeichen abschneidet und umgekehrt aneinandersetzt. Dies wird von folgendem kleinen Programm erledigt:

```
1000 REM -----------------------------------
1010 REM --- SPRITE AN X-ACHSE          ---
1020 REM ---                SPIEGELN ---
1030 REM -----------------------------------
1040 :
1050 SPRSAV 1,O$
1051 K$=""
1060 :
1070 FOR I=61 TO 1 STEP -3
1080 K$=K$+MID$(O$,I,3)
1090 NEXT
1100 :
1110 K$=K$+RIGHT$(O$,4)
1120 :
1130 SPRSAV K$,3
1140 SPRSAV K$,4
1150 :
1160 END
1170 :
```

Listing 4/6.1.1.1–1

Zuerst wird Sprite #1 mittels der Variablen O$ (für Original) in eine Zeichenreihe verwandelt und anschließend die Hilfsvariable K$ (für Kopie) als leere Zeichenreihe definiert. Dann werden in der Schleife ab Zeile 1070, beginnend ab dem 61. Zeichen, jeweils Zeichenreihen mit je drei Zeichen von hinten abgeschnitten und an die Variable K$ angefügt. Anschließend müssen zur Definition der Größe noch in Zeile 1110 die letzten vier Byte an K$ angefügt werden, womit das umgekehrte Sprite mit einer neuen Spritenummer versehen, gespeichert werden kann. Wir haben hier die Umkehrung in den beiden Spritenummern 3 und 4 abgelegt.

Wie Sie sich nun mit Hilfe des Sprite-Editors (SPRDEF) ansehen können, haben Sie nun in den Spritenummern 3 und 4 zwei im Kopfstand fliegende Luftschiffe.

Als nächstes wollen wir nun an der Y-Achse spiegeln und die Flugzeuge ‚rückwärts' fliegen lassen.

```
1000 REM ----------------------------------
1010 REM --- SPRITE AN Y-ACHSE          ---
1020 REM ---                SPIEGELN ---
1030 REM ----------------------------------
1040 :
1050 SPRSAV 1,O$
1060 K$=""
1070 :
1080 FOR I=1 TO 61 STEP 3
1090 :
1100 EI$=MID$(O$,I+2,1)
1110 GOSUB 2000
1120 K$=K$+AU$
1130 :
1140 EI$=MID$(O$,I+1,1)
1150 GOSUB 2000
1160 K$=K$+AU$
1170 :
1180 EI$=MID$(O$,I,1)
1190 GOSUB 2000
1200 K$=K$+AU$
1210 :
1220 NEXT
1230 :
1240 K$=K$+RIGHT$(O$,4)
1250 SPRSAV K$,5
1260 SPRSAV K$,6
1270 :
1280 END
1290 :
2000 REM ----------------------------------
2010 REM --- UP: BYTE SPIEGELN         ---
2020 REM ----------------------------------
2030 :
2040 EI=ASC(EI$)
2050 AU=0
2060 :
```

Listing 4/6.1.1.1-2 (1)

```
2070 IF EI AND 128 THEN AU=AU+  1
2080 IF EI AND  64 THEN AU=AU+  2
2090 IF EI AND  32 THEN AU=AU+  4
2100 IF EI AND  16 THEN AU=AU+  8
2110 IF EI AND   8 THEN AU=AU+ 16
2120 IF EI AND   4 THEN AU=AU+ 32
2130 IF EI AND   2 THEN AU=AU+ 64
2140 IF EI AND   1 THEN AU=AU+128
2150 :
2160 AU$=CHR$(AU)
2170 :
2180 RETURN
2190 :
2200 END
```

Listing 4/6.1.1.1–2 (2)

Hier liegt der Fall etwas komplizierter, da man nicht das linke mit dem rechten Byte vertauschen kann, sondern auch noch die Bits innerhalb der Bytes umgekehrt dargestellt werden müssen. Ebenso muß beim mittleren Byte einer Zeile eine Spiegelung innerhalb dieses Bytes erfolgen.

Besprechen wir zunächst das Unterprogramm zum Spiegeln eines einzelnen Bytes. Eingabeparameter ist die Variable EI$, die das zu spiegelnde Zeichen (Bitmuster) enthält und zur Umsetzung in die Variable EI als Zahl verwandelt wird. Als Ausgabe soll entsprechend eine Zeichenreihevariable AU$ fungieren, die das Zeichen (Bitmuster) gespiegelt enthält.

Zum Spiegeln werden die Zahlen herangezogen, also EI für das originäre Muster und AU für das gespiegelte. Die einfachste Möglichkeit besteht darin, jedes einzelne Bit im originären Muster abzufragen und das entsprechende Bit im gespiegelten Muster zu setzen, was die Zeilen 2070 bis 2140 für jedes einzelne Bit erledigen. Hierbei muß man sich veranschaulichen, daß den einzelnen Bits die 2er-Potenzen von 1 bis 128 zugeordnet sind. Wenn also in einem originären Bitmuster das erste Bit, entsprechend dem dezimalen Wert 128, gesetzt ist, so muß im gespiegelten Muster das letzte Bit, entsprechend dem Wert 1, gesetzt werden.

Das Bitmuster für die Ausgabe ergibt sich dann aus der Summe der addierten Zahlen. Weil die Umsetzung durch die Wahl der Dezimalzahlen bitweise erfolgt, und für jedes einzelne Bit eine eigene Bedingung kreiert wurde, braucht in Zeile 2160 lediglich noch das Zeichen zum entsprechenden ASCII-Code erstellt zu werden.

Bei dieser Vorgehensweise sollten Sie sich immer veranschaulichen, daß alle Informationen im Rechner nur in Bits (0/1) gespeichert sind und acht Bits zu einem Byte zusammengefaßt werden. Ob dieses Byte nun als Zahl zwischen 0 und 225 oder als Bitmuster oder auch als Zeichen aufgrund des ASCII-Codes (ASCII = American Standard Code of Information Interchange) aufgefaßt wird, ist Sache des Anwenders.

Das Hauptprogramm beginnt zunächst, ebenso wie bei unserem letzten Beispielprogramm, in dem die Daten aus Sprite #1 an die Variable O$ übergeben werden und die Zeichenreihe für das kopierte Zeichen geleert wird.

Im Gegensatz zum letzten Beispielprogramm beginnen wir hier jedoch beim ersten Zeichen, wobei jeweils drei Zeichen innerhalb einer Schleife behandelt werden. Als erstes muß das rechte Byte einer Zeile transformiert werden, um die transformierten Zeichen wiederum einfach an bereits transformierte in der Variablen K$ anzuhängen. Dies wird erreicht, indem wir in Zeile 1100 zum Wert der Laufvariablen noch ‚2' addieren und dann das soeben beschriebene Unterprogramm aufrufen. Dann muß das mittlere Zeichen innerhalb einer Sprite-Zeile transformiert werden, was in den Zeilen 1140 bis 1160 geschieht. Zum Schluß das erste Zeichen entsprechend in den Zeilen 1180 bis 1200.

Ebenfalls wie beim letzten Programm müssen die letzten vier Byte an die Variable K$ angehangen werden und dann kann die Zeichenreihe einer Spritenummer übergeben werden. Wir haben in unserem Beispiel die beiden Sprites 5 und 6 gewählt.

Achten Sie bei einer eigenen Anwendung darauf, daß Sie die Spritenummern entsprechend Ihren Wünschen ändern und vor dem Programmlauf auch das Originalsprite in den Hauptspeicher geladen ist.

Wenn Sie nun das gleiche Programm laufen lassen, als Originalsprite aber die Daten von Sprite #3 heranziehen und die Konvertierung in den Sprites #7 und #8 speichern, so haben Sie auch die zurückfliegenden Tiefflieger, indem Sie nacheinander an der X- und Y-Achse gespiegelt haben.

Das gleiche Verfahren können Sie auch mit dem SSHAPE-Befehl erstellten Variablen anwenden, wenn Sie sich die Werte in den letzten vier Byte zunutze machen und entsprechend als Anweisung bei den Laufvariablen einbauen.

4/6.2

Allgemeine Algorithmen

4/6.2.1

Sortierverfahren

Sortieren ist ein zentrales Thema im Zusammenhang mit Computern. Nachdem die Computer immer weniger zum Rechnen und immer mehr zur reinen Datenverarbeitung herangezogen werden, hat sich die Bedeutung des Sortierens von Daten auch weiter vergrößert.

Eine der drei Vorteile eines Computers im kommerziellen Einsatz, aber auch gelegentlich im Heimbereich, ist der Informationsgewinn. Gleiche Daten können auf vielfältige Weise verknüpft und insbesondere sortiert werden. Der Informationsgewinn resultiert aus der Schnelligkeit des Computers beim Erstellen und Drucken von Listen, die in Kleinbetrieben vorher nie erstellt wurden.

Dazu einige Beispiele: Wird bei einer Lagerverwaltung die umgesetzte Stückzahl bzw. der umgesetzte Geldwert gespeichert, und man sortiert nach diesen Werten, so kann man Hits und Ladenhüter sehr leicht lokalisieren. Bei einem umfangreichen Artikelsortiment eine Liste sortiert nach Stückumfang per Hand zu erstellen, hätte überhaupt keinen Sinn, da die Liste schon nicht mehr aktuell wäre, wenn sie fertig ist. Auch Karteileichen lassen sich mit Hilfe des Computers schnell auffinden, wenn man z.B. in einen Kundendatensatz einen Eintrag 'Datum der letzten Rechnung' vorsieht.

Ebenso bietet das Sortieren bei einer privaten Adreßdatei Vorteile. Bei normaler schriftlicher Notierung in einem kleinen Buch werden alle Adressen unter ihrem Anfangsbuchstaben, und dort nicht weiter sortiert, eingeordnet. Selbst wenn man zu jeder Adresse das Geburtsdatum dazu schreibt, vergißt man so manchen Geburtstag, wenn man sich nicht eine extra Geburtstagsliste, eventuell in einem Terminkalender, anlegt. Sind alle Adressen inklusive Geburtsdatum erfaßt, so ist es eine Kleinigkeit für den Computer, eine Liste der Adressen, nach Kalenderdatum geordnet, aufzustellen.

Aufgrund der vielfältigen Einsatzmöglichkeiten von Sortierverfahren ist es nicht weiter verwunderlich, wenn im Laufe der Zeit immer bessere und schnellere Sortierverfahren entwickelt wurden. Wir wollen in diesem Kapitel nicht alle Sortierverfahren besprechen, da alle Verfahren ihre Vor- und Nachteile haben, aber drei wichtige Verfahren herauspicken, die für die verschiedensten Einsatzzwecke von Vorteil sind. Dies ist zum einen das eigentlich langsamste Sortierverfahren, das aber bei bereits vorsortierten Daten eine enorme Schnelligkeit entwickelt, zum anderen ein mittelschnelles Sortierverfahren, das sehr leicht programmierbar ist, welches wir für auf- und absteigendes Sortieren vorstellen, und natürlich das zur Zeit schnellste Sortierprogramm überhaupt, was man auch seinem Namen entnehmen kann: Quicksort.

SHELL-SORT

Die Vorgehensweise bei den drei verschiedenen Sortierverfahren beschreiben wir am besten durch einen Vergleich mit einem Stapel zu sortierender Karteikarten, wobei der Stapel Karteikarten natürlich nicht vorsortiert ist.

Das Verfahren SHELL-SORT ist dabei leicht erklärt. Sie fangen vorne bei den Karteikarten an und vergleichen immer zwei aufeinanderfolgende Karteikarten. Wenn der Eintrag auf der hinteren Karteikarte kleiner ist als der Eintrag auf der vorderen Karteikarte, werden einfach beide Karten vertauscht. Die hintere Karteikarte bei der aktuellen Durchsuchung ist gleichzeitig die vordere Karteikarte für die nächste Durchsuchung. Wenn Sie am Ende angekommen sind, fangen Sie wieder von vorne an. Das Sortieren ist abgeschlossen, wenn Sie einen Durchlauf durch den Stoß Karteikarten gemacht haben und keine einzige Vertauschung durchgeführt wurde.

Durch dieses Vertauschen werden die Karteikarten mit den kleineren Einträgen langsam nach vorne bewegt, bis sie ihren Platz erreicht haben, und gleichzeitig Karteikarten mit alphanumerisch großen Einträgen nach hinten. Der Aufwand für dieses Sortierverfahren ist natürlich sehr groß, was Sie schon alleine an einem Spezialfall sehen. Alle Karteikarten sind sortiert bis auf eine, die kleinste, die sich am Schluß befindet. Bei 1000 Karteikarten müssen sie 999mal den Stapel durchgehen, bis die kleinste Karte endlich vorne ist. Dies bedeutet ca. 1.000.000 Überprüfungen und 999 Vertauschungen. Dies ist gleichzeitig auch der Extremfall (von Überprüfungen), da hier der längste Weg für eine Karteikarte zurückzulegen ist. Wären auch noch andere Karten nicht an der richtigen Stelle, würden sie im Laufe der Durchgänge automatisch mit nach vorne wandern.

Da Sortierprogrammteile in vielen Anwenderprogrammen benötigt werden, haben wir die Beschreibung den Unterprogrammen aus Kapitel 4/4.7 angepaßt und etwas ausführlicher gestaltet. Zunächst also die Dokumentation für SHELL-SORT zum Sortieren von Zahlen (die Programmierung für Sortierung nach Zeichenreihen benutzt nur Stringvariablen), mit dem ein Feld WE ( ) sortiert wird.

Name: **SHELLZAHL**
Zweck: **SHELL-SORT für Zahlen**

```
1000 REM -------------------------------
1010 REM - TESTRAHMEN FUER SHELL-SORT -
1020 REM -------------------------------
1030 :
1040 AN=15
1050 DIM WE(AN)
1060 :
1070 FOR I=1 TO AN
1080 : WE(I)=INT(RND(TI)*1000)+1
1090 NEXT
1100 :
1110 T=TI
1120 GOSUB 7000
1130 T=(TI-T)/60
1140 :
1150 FOR I=1 TO AN
1160 : PRINT I,WE(I)
1170 NEXT
1180 :
1190 PRINT"BENOETIGTE ZEIT ";T;"SEKUNDEN = ";T/60;"MINUTEN"
1200 :
1210 END
1220 :
7000 REM -------------------------------
7010 REM ---        SHELL-SORT        ---
7020 REM -------------------------------
7030 :
7040 FE=1
7050 :
7060 FOR I=1 TO AN-1
7070 : IF WE(I+1)<WE(I) THEN HI=WE(I) : WE(I)=WE(I+1) : WE(I+1)=HI : FE=0
7080 NEXT
7090 :
7100 IF FE=0 THEN 7040
7110 :
7120 RETURN
7130 :
```

Listing 4/6.2.1-1

Struktogramm:

Merker für sortiertes Feld setzen

Solange noch nicht fertig

- Für alle Werte des Feldes bis zum vorletzten
  - Ist nächstes Element im Feld größer als das aktuelle Feldelement?
    - ja:
      - vertausche beide Felder (mit Hilfsvariable HI)
      - setze Merker für sortiertes Feld zurück
    - nein

Rücksprung

Variablen:

Name	zul. Bereich	Bedeutung
Eingabeparameter:		
AN WE( )	Integer Dezimalzahlen	Anzahl der Elemente im zu sortierenden Feld zu sortierendes Feld
Ausgabeparameter:		
WE( )	Dezimalzahlen	sortiertes Feld
Lokale Variablen:		
FE HI I	0, 1 Dezimalzahl 1 . . . AN-1	Merker für fertig sortiertes Feld Hilfsvariablen zum Vertauschen zweier Feldelemente Laufvariable
Globale Variablen:		keine

Rücksprünge:

In Zeile	Art	Bedingung	Bemerkung
7120	Return	—	fertig

Aufgrund der zu Anfang gegebenen Beschreibung der Arbeitsweise des Sortierverfahrens und der Dokumentation, sowie den auf den nächsten Seiten aufgeführten Beispielen, erübrigt sich jede weitere Erklärung. Da SHELL-SORT unser erstes Sortierprogramm ist, wollen wir hier auch das Listing für Sortierung nach Zeichenreihen aufzeigen, bei dem — neben dem geänderten Testrahmen — lediglich die Variablennamen zu ändern sind.

```
1000 REM -------------------------------
1010 REM - TESTRAHMEN FUER SHELL-SORT -
1020 REM -------------------------------
1030 :
1040 AN=15
1050 DIM WE$(AN)
1060 :
1070 DATA INTEREST,MAIER,MEIER,MAYER,MAYR,MEYER,SCHNEIDER,SCHMIDT,SCHMITT
1080 DATA SCHMITZ,NIGGL,EBERL,FRIESE,STRAUSS,WIMMER
1090 :
1100 FOR I=1 TO AN
1110 : READ WE$(I)
1120 NEXT
1130 :
1140 T=TI
1150 GOSUB 7500
1160 T=(TI-T)/60
1170 :
1180 FOR I=1 TO AN
1190 : PRINT I,WE$(I)
1200 NEXT
1210 :
1220 PRINT"BENOETIGTE ZEIT ";T;"SEKUNDEN = ";T/60;"MINUTEN"
1230 :
1240 END
1250 :
7500 REM ------------------------------
7510 REM ---        SHELL-SORT      ---
7520 REM ------------------------------
7530 :
7540 FE=1
7550 :
7560 FOR I=1 TO AN-1
7570 : IF WE$(I+1)<WE$(I) THEN HI$=WE$(I) : WE$(I)=WE$(I+1) : WE$(I+1)=HI$ : FE=
0
7580 NEXT
7590 :
7600 IF FE=0 THEN 7540
7610 :
7620 RETURN
7630 :
```

Listing 4/6.2.1-2

Lassen Sie uns abschließend noch kurz auf das Zeitverhalten und einige Verbesserungen bei SHELL-SORT eingehen. Bei 10 unsortierten Zahlen braucht SHELL-SOIRT ca. zwei Sekunden zum Sortieren. Bei 50 unsortierten Zahlen werden bereits ca. 47 Sekunden benötigt, bei 100 Zahlen ca. 3 Minuten und 11 Sekunden. Weitere Daten entnehmen Sie bitte der Gegenüberstellung am Schluß des Kapitels.

Für große zu sortierende Mengen ist SHELL-SORT anscheinend nicht das geeignete Hilfsmittel. Aber SHELL-SORT ist das beste Hilfsmittel um zu prüfen, ob eine sortierte Folge vorliegt. Hier die ungefähren Zeitangaben bei sortierten Daten: 0,5 Sekunden (für 10 sortierte Zahlen), 1,5 Sekunden (für 100 Zahlen), 15,5 Sekunden (1000) und nur ca. 77 Sekunden für 5000 sortierte Zahlen, im FAST-Modus des C 128 sogar nur 37 Sekunden. In diesem Fall ist das vorgestellte Sortierverfahren unschlagbar. Anwendungsbeispiele für das Prüfen, ob sortierte Daten vorliegen, sind sehr vielfältig. Wenn man die unten beschriebenen Verfahren zur Beschleunigung von SHELL-SORT anwendet, und unser Extrembeispiel vom Beginn heranzieht, ergibt sich auch ein gutes Einsatzfeld im Heimbereich: Sie haben 1000 Adressen, Schallplatten oder Dias gespeichert, die sortiert vorliegen. Wenn Sie nur wenige Daten neu erfassen müssen, wäre es unsinnig, alle Daten vollkommen neu zu sortieren. Sie brauchen lediglich an die richtige Stelle verschoben zu werden, wobei SHELL-SORT auch wieder Vorteile für sich verbuchen kann.

Es wurden einige Wege gefunden, um SHELL-SORT etwas schneller zu machen, von denen wir hier zwei kurz erläutern wollen. Zum einen kann man sich die jeweiligen äußeren Grenzen merken, wo die Karteikarten schon sortiert sind. Nehmen wir an, Sie stellen beim ersten Durchlauf fest, bis zur 500. und ab der 700. Karteikarte ist der Stapel bereits richtig sortiert, so ist nur der Bereich von 500 bis 700 zu durchsuchen, wobei es allerdings sein kann, daß die Grenzen sich nach außen hin verschieben, Sie also das nächstemal von 499 bis 701 durchsuchen müssen. Für unser eingangs erwähntes Extrembeispiel (alles sortiert, nur das kleinste zu sortierende Element befindet sich am Ende) würde dies bedeuten, daß wir beim ersten Durchlauf feststellen, daß die Elemente Nummer 999 und 1000 zu vertauschen sind, das Element mit der Nummer 1000 dann richtig ist, und wir den zweiten Durchlauf bei 998 beginnen und bei 999 beenden können. Die Vertauschungen werden dann alle direkt ausgeführt, ohne daß wir sämtliche Elemente bei jedem Durchlauf überprüfen müssen.

Damit sind wir auch gleich bei der zweiten Möglichkeit, das Verfahren SHELL-SORT zu beschleunigen: Sobald wir beim Prüfen feststellen, daß zwei Karten vertauscht werden müssen, gehen wir nicht nach hinten weiter, sondern prüfen die vordere der vertauschten Karten (die ja gerade nach vorne gewandert ist) mit der Karte davor.

Mit anderen Worten, es wird eine weiter hinten gefundene kleinere Karteikarte solange nach vorne vertauscht, bis sie ihren richtigen Platz hat. Dann kann man an der Stelle weitermachen, wo die erste Vertauschungsserie begonnen hat.

Abschließend noch drei kleine, einfache Beispiele, die die unterschiedlichen Vorgänge deutlich machen (p bedeutet prüfen; pv bedeutet prüfen und vertauschen):

SHELL-SORT normal **Schritt**

```
10      20      30      45      14      78      85      65        ( 1)
!______!  p
10      20      30      45      14      78      85      65        ( 2)
        !______!  p
10      20      30      45      14      78      85      65        ( 3)
                !______!  p
10      20      30      45      14      78      85      65        ( 4)
                        !______!  pv
10      20      30      14      45      78      85      65        ( 5)
                                !______!  p
10      20      30      14      45      78      85      65        ( 6)
                                        !______!  p
10      20      30      14      45      78      85      65        ( 7)
                                                !______!  pv
10      20      30      14      45      78      65      85        ( 8)
!______!  p
10      20      30      14      45      78      65      85        ( 9)
        !______!  p
10      20      30      14      45      78      65      85        (10)
                !______!  pv
10      20      14      30      45      78      65      86        (11)
                        !______!  p
10      20      14      30      45      78      65      85        (12)
                                !______!  p
10      20      14      30      45      78      65      85        (13)
                                        !______!  pv
10      20      14      30      45      65      78      85        (14)
                                                !______!  p
10      20      14      30      45      65      78      85        (15)
!______!  p
10      20      14      30      45      65      78      85        (16)
        !______!  pv
10      14      20      30      45      65      78      85        (17)
                !______!  p
10      14      20      30      45      65      78      85        (18)
                        !______!  p
10      14      20      30      45      65      78      85        (19)
                                !______!  p
10      14      20      30      45      65      78      85        (20)
                                        !______!  p
10      14      20      30      45      65      78      85        (21)
                                                !______!  p
```

Man benötigt sieben weitere Schritte, um festzustellen, daß man fertig ist, für acht — fast sortierte — Zahlen also 28 Schritte.

SHELL-SORT mit Grenzen (U: untere Grenze, O: obere Grenze)

```
10        20        30        45        14        78        85        65        ( 1)
!___________! p
10        20        30        45        14        78        85        65        ( 2)
          !___________! p
10        20        30        45        14        78        85        65        ( 3)
                    !___________!  p
10        20        30        45        14        78        85        65        ( 4)
                              !___________! pv=U
10        20        30        14        45        78        85        65        ( 5)
                                        !___________! p
10        20        30        14        45        78        85        65        ( 6)
                                                  !___________! p
10        20        30        14        45        78        85        65        ( 7)
                                                            !___________! pv=O
10        20        30        14        45        78        65        85        ( 8)
                    !___________! pv=U
10        20        14        30        45        78        65        85        ( 9)
                              !___________! p
10        20        14        30        45        78        65        85        (10)
                                        !___________! p
10        20        14        30        45        78        65        85        (11)
                                                  !___________! pv=O
10        20        14        30        45        65        78        85        (12)
          !___________! pv=U
10        14        20        30        45        65        78        85        (13)
                    !___________! p
10        14        20        30        45        65        78        85        (14)
                              !___________! p
10        14        20        30        45        65        78        85        (15)
                                        !___________! p               *         .
10        14        20        30        45        65        78        85        (16)
!___________! p=U=O
```

Wie Sie an diesem Beispiel sehen, ist prinzipiell beim Verrücken der Untergrenze eine Einheit vorher mit der Prüfung zu beginnen und bei Erreichen der Obergrenze kann eine Einheit früher aufgehört werden. Da in den Schritten 13 bis 15 keine Vertauschungen mehr vorgenommen werden, wird an der Stelle * festgestellt, daß die Untergrenze gleich der Obergrenze ist, und in Schritt 16 stellt man fest, daß die beiden Grenzen nicht weiter verändert werden brauchen.

Trotz der ungünstigen Obergrenze benötigen wir nur 16 Schritte, fast die Hälfte der 28 Schritte beim 'normalen' SHELL-SORT.

SHELL-SORT mit fortlaufender Vertauschung
(M bedeutet Position merken)

```
10     20     30     45     14     78     85     65     ( 1)
!______! p
10     20     30     45     14     78     85     65     ( 2)
       !______! p
10     20     30     45     14     78     85     65     ( 3)
              !______! p
10     20     30     45     14     78     85     65     ( 4)
                     !______! pv=M
10     20     30     14     45     78     85     65     ( 5)
              !______! pv
10     20     14     30     45     78     85     65     ( 6)
       !______! pv
10     14     20     30     45     78     85     65     ( 7)
!______! p
10     14     20     30     45     78     85     65     ( 8)
                     !______! p (gemerkte Stelle)
10     14     20     30     45     78     85     65     ( 9)
                            !______! p
10     14     20     30     45     78     85     65     (10)
                                   !______! p
10     14     20     30     45     78     85     65     (11)
                                          !______! pv=M
```

Die gemerkte Stelle ist hier bereits das Ende, so daß nach abgeschlossener Vertauschung der Sortiervorgang beendet ist.

```
10    20    14    30    45    78    65    85    (12)
                            !_____! pv

10    14    20    30    45    65    78    85    (13)
                      !_____! p
```

Selbst in unserem — aus Platzgründen recht einfach gewählten Beispiel — sparen wir in der letzten Version nochmals drei Schritte ein.

Weil alphanumerisch kleinere Karteikarten so nach vorne wandern, wie Blasen (englisch Bubbles) in einer Flüssigkeit nach oben steigen, wird das Verfahren auch als Bubble-Sort bezeichnet.

MINSORT/MAXSORT

Das nächste Verfahren, das wir vorstellen wollen, ist etwas schneller, hat aber seinen besonderen Vorteil in der relativ einfachen Programmierung gegenüber dem noch schnelleren Verfahren QUICKSORT. Je nachdem, ob man aufsteigend oder absteigend sortieren will, wird das Verfahren als MINSORT (von **MIN**imum) oder MAXSORT (von **MAX**imum) bezeichnet.

Auch hier wollen wir uns wieder einen Stapel unsortierter Karteikarten vorstellen. Da die Beschreibung für auf- bzw. absteigende Sortierung ähnlich ist — es kommt nachher nur darauf an, von welcher Seite man sich den Stoß ansieht — besprechen wir hier die aufsteigende Sortierung.

Der Vorgang: Sie durchsuchen den Stoß von vorne nach hinten, wobei Sie sich jeweils die Karteikarte mit dem kleinsten Eintrag merken, eventuell durch leichtes Herausziehen dieser Karteikarte. Zu Beginn wird die erste Karte als kleinstes Element herangezogen und jede der folgenden Karten vergleichen Sie mit der ersten Karte, bis Sie eine kleinere finden, die dann markiert und zum Vergleich herangezogen wird (die vorher als Kleinere ausgesuchte verschwindet wieder im Stapel). Haben Sie den ganzen Stoß durchsucht, entnehmen Sie einfach die markierte kleinste Karte und legen sie umgedreht neben den Stoß. Ebenso verfahren Sie in den nächsten Durchgängen und legen alle anderen gefundenen Karten des Stapels nacheinander auf den neuen Stapel. Als letztes werden Sie dann die Karteikarte mit dem größten Eintrag in der Hand halten.

Wenn Sie fertig sind, haben Sie z.B. bei 1000 Karteikarten 500.000 Überprüfungen vorgenommen. Allgemein sind $n^2/2$ Überprüfungen notwendig, da bei fortschreitender Suche der zu durchsuchende Stapel immer kleiner wird.

Weil wir mit zwei Stapeln operieren, müßten wir im Computer auch zwei Felder vorsehen, das eine zum Ablegen der gefundenen jeweiligen kleinsten Elemente, und das andere Feld mit den zu sortierenden Daten, wobei ein aussortierter Datensatz natürlich noch kenntlich gemacht werden muß.

Da sich die Anzahl der Karteikarten naturgemäß beim Sortieren nicht ändert, läßt sich die Sortierung allerdings auch in einem einzigen Feld erledigen. Dazu sind — wie bei SHELL-SORT — nach einem Durchsuchungsvorgang des restlichen Stapels zwei Karten zu vertauschen und zwar die vorderste vom zu durchsuchenden Reststapel mit der Karte mit dem kleinsten gefundenen Eintrag. Die jeweiligen Karten mit den kleinsten Einträgen werden nicht mehr auf einen gesonderten Stoß gelegt, sondern nach vorne geholt.

Mit Kenntnis dieser Vorgehensweise ist es für Sie sicherlich nicht schwierig, daraus ein Computerprogramm zu bilden. Auch hier wollen wir wieder das eigentliche Sortieren als Unterprogramm ausgestalten, so daß es von jedem Anwenderprogramm unter Berücksichtigung der Ein-/Ausgabeparameter verwendet werden kann.

Name: **MINZAHL**
Zweck: **MINSORT für Zahlen**

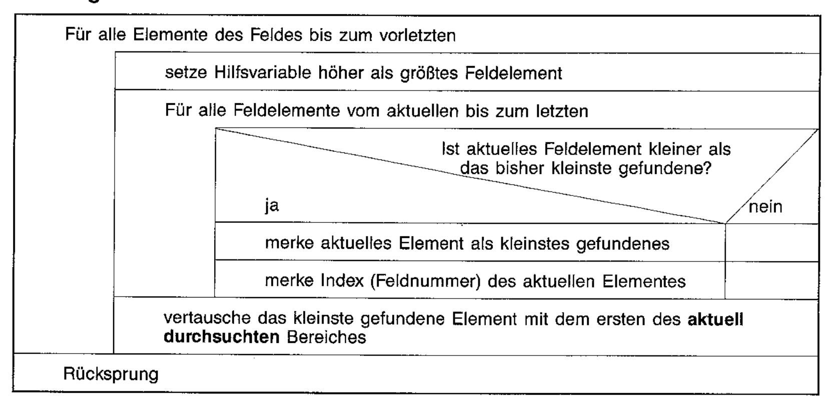
```
1000 REM ------------------------------
1010 REM --- TESTRAHMEN FUER MINSORT --
1020 REM ------------------------------
1030 :
1040 AN=15
1050 DIM WE(ANZAHL)
1060 :
1070 FOR I=1 TO AN
1080 : WE(I)=INT(RND(TI)*1000)+1
1090 NEXT
1100 :
1110 T=TI
1120 GOSUB 8000
1130 T=(TI-T)/60
1140 :
1150 FOR I=1 TO AN
1160 : PRINT I,WE(I)
1170 NEXT
1180 :
1190 PRINT"BENOETIGTE ZEIT ";T;"SEKUNDEN = ";T/60;"MINUTEN"
1200 :
1210 END
1220 :
8000 REM ------------------------------
8010 REM ---         MINSORT         ---
8020 REM ------------------------------
8030 :
8040 FOR I=1 TO AN-1
8050 : MI=1000000
8060 :
8070 : FOR J=I TO AN
```

Listing 4/6.2.1-3 (Teil 1)

```
8080 : : IF WE(J) < MI THEN MI=WE(J) : WO=J
8090 : NEXT
8100 :
8110 : HI=WE(I) : WE(I)=WE(WO) : WE(WO)=HI
8120 NEXT
8130 :
8140 RETURN
8150 :
```

Listing 4/6.2.1-3 (Teil 2)

Struktogramm:



Variablen:

Name	zul. Bereich	Bedeutung
Eingabeparameter:		
A	Integer	Anzahl der Elemente im zu sortierenden Feld
WE( )	Dezimalzahlen	zu sortierendes Feld

Ausgabeparameter:		
WE( )	Dezimalzahl	sortiertes Feld
Lokale Variablen:		
HI	Dezimalzahl	Hilfsvariablen zum Vertauschen zweier Feldelemente
I	1 . . .AN-1	Laufvariable
J	i . . . AN	Laufvariable
MI	. . . 1000 000	Merker für kleinstes gefundenes Element
WO	1 . . .AN	Merker für Index des kleinsten gefundenen Elementes
Globale Variablen:		keine

Rücksprünge:

In Zeile	Art	Bedingung	Bemerkung
8140	Return	—	fertig

Da es sich beim alphanumerischen Sortieren nur um kleine Änderungen handelt, die wir bei SHELL-SORT schon besprochen haben, soll auf die Abbildung der Programmänderungen und des Testprogrammes verzichtet werden.

Bei MINSORT (aufsteigendes Sortieren) stellt die äußere Schleife (Laufvariable: i) jeweils den zu durchsuchenden Restteil dar, wobei das erste Element dieses Teiles nach Abschluß des Suchens mit dem gefundenen, kleinsten Element vertauscht wird. Die innere Programmschleife ist dabei für das eigentliche Durchsuchen zuständig, und unsere Hilfsvariable WO stellt das Markieren des gefundenen kleinsten Elementes dar. Wenn Sie das Feldelement WE(0) nicht benutzen, kann statt der Hilfsvariable HI auch das Element WE(0) zum Vertauschen herangezogen werden.

Das aufsteigende Sortieren (MAXSORT) kann man auf zwei Wegen erreichen: Einerseits, indem man immer das größte Element sucht und nach vorne bringt, andererseits, indem man weiterhin das kleinste Element sucht, dies aber — bei unserem bildlichen Beispiel — hinten in den Stapel einbringt. Wir haben uns in unserem Programm für die letzte Möglichkeit entschieden, und so läuft unsere äußere Schleife von hinten nach vorne und die innere Schleife nicht vom aktuellen Element 'i' bis zum Ende, sondern vom Anfang bis zu diesem Element. Da ansonsten die Dokumentation derjenigen von MINSORT entspricht, möchten wir Ihnen lediglich das Programm zum Sortieren von Zahlen, sowie das zugehörige Testprogramm zum direkten Abtippen darstellen.

```
1000 REM ------------------------------
1010 REM --- TESTRAHMEN FUER MAXSORT --
1020 REM ------------------------------
1030 :
1040 AN=10
1050 DIM WE(AN)
1060 :
1070 FOR I=1 TO AN
1080 : WE(I)=INT(RND(TI)*1000)+1
1090 NEXT
1100 :
1110 T=TI
1120 GOSUB 8500
1130 T=(TI-T)/60
1140 :
1150 FOR I=1 TO AN
1160 : PRINT I,WE(I)
1170 NEXT
1180 :
1190 PRINT"BENOETIGTE ZEIT ";T;"SEKUNDEN = ";T/60;"MINUTEN"
1200 :
1210 END
1220 :
8500 REM ------------------------------
8510 REM ---        MAXSORT          ---
8520 REM ------------------------------
8530 :
8540 FOR I=AN TO 2 STEP -1
8550 : MI=1000000
8560 :
8570 : FOR J=1 TO I
8580 : : IF WE(J) < MI THEN MI=WE(J) : WO=J
8590 : NEXT
8600 :
8610 : HI=WE(I) : WE(I)=WE(WO) : WE(WO)=HI
8620 NEXT
8630 :
8640 RETURN
8650 :
```

Listing 4/6.2.1-4

Ähnlich wie bei SHELL-SORT wollen wir ein kures Ablaufbeispiel durchrechnen:

10	20	30	47	85	96	87	54	12	65	91	58	59	63	75
10	12	30	47	85	96	87	54	20	65	91	58	59	63	75
10	12	20	47	85	96	87	54	30	65	91	58	59	63	75
10	12	20	30	85	96	87	54	47	65	91	58	59	63	75
10	12	20	30	47	96	87	54	85	65	91	58	59	63	75
10	12	20	30	47	54	87	96	85	65	91	58	59	63	75
10	12	20	30	47	54	58	96	85	65	91	87	59	63	75
10	12	20	30	47	54	58	59	85	65	91	87	96	63	75
10	12	20	30	47	54	58	59	63	65	91	87	96	85	75
10	12	20	30	47	54	58	59	63	65	91	87	96	85	76
10	12	20	30	47	54	58	59	63	65	75	87	96	85	91
10	12	20	30	47	54	58	59	63	65	75	85	96	87	91
10	12	20	30	47	54	58	59	63	65	75	85	87	96	91
10	12	20	30	47	54	58	59	63	65	75	85	87	91	96

Das letzte Element braucht offensichtlich nicht mehr sortiert zu werden.

Auch hier wollen wir wieder einige kurze Zeitvergleiche anstellen, zunächst wieder für unsortierte Zahlen. MINSORT/MAXSORT brauchen für 10 Zahlen etwa 1/2 Sekunde, für 50 Zahlen knapp 8 Sekunden, für 100 Zahlen ungefähr 28 Sekunden und für 500 Zahlen circa 10,5 Minuten, was bei 500 Zahlen in etwa einem Viertel der Rechenzeit von SHELL-SORT ausmacht. Im Gegensatz zu SHELL-SORT verkürzen sortierte Daten die Rechenzeit nicht, wie man aufgrund des Verfahrensablaufes schon ablesen kann, da immer alle Elemente durchsucht werden müssen.

Bei sortierten Daten wäre es vielleicht sinnvoll, das Vertauschen vorher noch durch eine Bedingung zu prüfen, ob überhaupt vertauscht werden muß. Bei unsortierten Daten würde diese Bedingungsabfrage jedoch die Rechenzeit noch erhöhen und bei sortierten Daten nur geringe Zeitersparnis bringen.

Der Vorteil von MINSORT/MAXSORT liegt in der relativ einfachen Programmierung und ist ca. 3 bis 4 mal schneller als SHELL-SORT. Für nicht zu große Datenmengen reicht dieses Sortierverfahren aus, und so werden wir es auch als Hilfssortierverfahren für das nächste Verfahren verwenden.

QUICKSORT

QUICKSORT ist das derzeit schnellste bekannte Verfahren für unsortierte Datenmengen, wobei sortierte Daten allerdings auch in der Rechenzeit berücksichtigt werden.

Erläutern wir das Verfahren wieder an einem Stapel mit 1000 Karteikarten, die z.B. nach Namen sortiert werden sollen. Das Verfahren QUICKSORT ist deshalb so kompliziert, weil es in einem Durchgang nur eine gewisse Teilsortierung herstellt, und man die teilsortierten Bereiche anschließend wieder mit einem beliebigen Sortierverfahren weitersortieren kann, z.B. auch QUICKSORT. Da QUICKSORT sich für Teilbereiche immer wieder selber aufrufen kann, ist es auch ein sogenanntes rekursives Sortierverfahren. Bei Basic stellt sich natürlich das Problem der rekursiven Programmierung, da rekursive Unterprogramme — ähnlich PASCAL — nicht vorgesehen sind. Bei unserer bildlichen Darstellung werden Sie diese Rekursivität sehr schnell erkennen.

Sie sollten bei der gedanklichen Bearbeitung der Karteikarten schon einmal etwas Platz um sich herum schaffen, denn Sie werden ihn brauchen.

Zunächst sucht man sich bei QUICKSORT ein Vergleichselement, das ungefähr in der Mitte der zu sortierenden Daten liegt, bei Karteikarten nach Namen z.B. ein Name, der mit N anfängt. Obwohl das folgende Verfahren für Karteikarten etwas außergewöhnlich anmutet, ist es sicherlich auch im praktischen Einsatz das schnellste — und kompliziertestе. Stellen Sie den Kartenstapel **quer** vor sich, so daß Sie links den vorderen und rechts den hinteren Teil des Kartenstapels bearbeiten können.

Nun beginnen sie links mit der Bearbeitung. Als Vergleichskriterium hatten wir einen Namen mit N gewählt. Wir legen die geprüften Karteikarten jeweils rechts und links von unserem Ausgangsstapel auf einen neuen Stapel. Alle Namen, die mit A bis M beginnen, gehören auf den linken Stoß und alle Namen von N bis Z auf den rechten Stoß. So lange die Karten, die sie von **links** dem Ursprungsstapel entnehmen, ihren Anfangsbuchstaben im Bereich A bis M haben, legen Sie diese einfach auf den linken Stoß. Sobald die erste Karte von dieser Seite einen Namen mit dem Anfangsbuchstaben aus dem Bereich N bis Z beinhaltet, legen Sie diese Karte zwar oben auf, beenden aber die Überprüfung auf der linken Seite. Die Überprüfung wird auf der **rechten** Seite fortgesetzt, und zwar werden jetzt alle Karten auf den rechten Stoß gelegt, die in dem Bereich N bis Z ihren Anfangsbuchstaben haben.

Sobald Sie auf die erste Karte treffen — wir sind immer noch rechts — deren Anfangsbuchstabe im Bereich A bis M liegt, legen Sie diese Karte ebenfalls noch obenauf und brechen den Prüfvorgang ab.

Sie haben jetzt neben den zu sortierenden Karteikarten links einen Stoß, wo alle Namen mit A bis M beginnen, mit Ausnahme der obersten Karte, und rechts einen Stapel, bei dem die Anfangsbuchstaben alle mit N bis Z anfangen, bis auf die oberste Karte. Wenn Sie nun die beiden obersten Karten vertauschen, gilt das eben Gesagte für beide Stöße, diesmal ohne Ausnahme.

Machen Sie nun links weiter und prüfen Sie, ob die Karten im Bereich A bis M liegen ,und brechen Sie die Überprüfung bei einem anderen Anfangsbuchstaben sofort ab. Daraufhin prüfen Sie den rechten Bereich wieder auf den Bereich N bis Z. Ist auch hier eine Karte aus dem falschen Bereich aufgetaucht, tauschen sie die beiden obersten Karten von den Stapeln rechts und links wieder aus und setzen die Prüfung fort, bis der ganze zu sortierende Kartenstoß auf die beiden Stapel verteilt ist.

Sie haben nun einen Stoß links, wo die Anfangsbuchstaben **ausnahmslos** im Bereich A bis M liegen und rechts einen Stoß für den Bereich N bis Z. Auf diese beiden Stöße können Sie nun ein beliebiges Sortierverfahren anwenden, gegebenenfalls für jeden Stoß ein anderes Sortierverfahren. Wenn wir mit 1000 Karten angefangen haben, haben wir jetzt auf jedem Stoß **ungefähr** 500 Karten, wenn die Namen auf den Karten nicht alle Schulz, Schmitt, Schmitz oder Schneider sind. Man sollte sein Vergleichskriterium (bei uns das N) so wählen, daß es möglichst zu einer gleichmäßigen Aufteilung kommt. Von der geschickten Wahl Ihres Vergleichskriteriums hängt entscheidend die gesamte Sortierzeit ab.

Wenden wir nun z.B. auf den Stapel mit den Anfangsbuchstaben N bis Z wieder das Verfahren QUICKSORT an, so müssen wir natürlich ein neues Vergleichskriterium suchen, z.B. 'T'. Ein Stapel wird nach erneuter Verteilung, wie oben beschrieben, gebildet, der alle Karteikarten mit den Anfangsbuchstaben N bis S enthält, und der andere die von T bis Z. Sie sollten immer den Stapel mit den größten Sortierkriterien zuerst behandeln. Als nächstes würde also der Stapel mit den Karten der Anfangsbuchstaben T bis Z behandelt. Zur weiteren Bearbeitung liegen dann immer noch die Stapel mit den Anfangsbuchstaben von A bis M und von N bis S.

Sie sehen, das Verfahren ist sehr platzaufwendig, da Sie für jeden zu bearbeitenden Stapel zwei neue erhalten. Wenn Sie auf jeden Stapel immer wieder QUICKSORT anwenden, müssen Sie dies so oft tun, bis jeder Stapel nur noch aus einer einzigen Karte besteht. Sicherlich wird Ihnen bis dahin der Platz ausgegangen sein, und Sie werden sich z.B. entschließen, bei 10 oder 15 Karten nicht mehr QUICKSORT als Sortierverfahren zu wählen, sondern z.B. SHELL-SORT oder MINSORT/MAX-SORT.

In Wirklichkeit würden Sie natürlich nicht die Stapel von links und rechts bearbeiten, sondern den Ursprungsstapel in die Hand nehmen und nach den erwähnten Kriterien auf zwei Stapel aufteilen, wie man es z.B. auch bei einen 104-Blatt-Kartenspiel durchführt, wenn die roten und blauen Karten wieder getrennt werden sollen. Die genannte Vorgehensweise erlaubt allerdings eine einfachere Übertragung für den Computer, da wir wieder innerhalb des Ausgangsfeldes sortieren können, und kein Hilfsfeld benötigen. Letztendlich wird es im Computer also bei einem Stapel bleiben, wobei der Rechner sich für die einzelnen Bereiche Trennelemente merkt.

Im Computer brauchen wir also nicht soviel Platz, da wir — wie bei MINSORT/MAXSORT — mit einigen kleinen Tricks ein Feld in sich selbst sortieren können. Bei Programmiersprachen mit rekursiver Programmiermöglichkeit, wie z.B. PASCAL, wird dabei die Grenzenverwaltung (in unseren Beispielen A bis M, N bis S, T bis Z) durch das Programm selbst übernommen.

Bei Basic müssen wir einen kleinen Umweg gehen und uns diese Grenzen selbst merken. Da das Merken und Aufbereiten der Grenzen in Basic verhältnismäßig lange dauert, ist es hier von Vorteil, bei kleinen 'Stapeln' auf ein anderes Verfahren auszuweichen, wozu wir MINSORT gewählt haben.

Bevor wir uns der eigentlichen Programmierung zuwenden, wollen wir wieder ein kleines Zahlenbeispiel vorführen, wo wir die später verwendeten Zeiger schon einbauen (p bedeutet 'nur prüfen' und pv bedeutet 'prüfen und später vertauschen'):

Ausgangssituation:

```
10  20  45  74  87  52  46  14  12  51  63  89  87  73  58  i= 1
!                                                       !  j=15
links=links(1)                                 rechts(1)=rechts
 =i                  Vergleichselement = 50              =j
                     Zeiger=0 (vor sortieren)
```

1. Durchlauf

10	20	45	74	87	52	46	14	12	51	63	89	87	73	58	i= 1
i p														j	j=15
10	20	45	74	87	52	46	14	12	51	63	89	87	73	58	i= 2
	i p													j	j=15
10	20	45	74	87	52	46	14	12	51	63	89	87	73	58	i= 3
		i p												j	j=15
10	20	45	74	87	52	46	14	12	51	63	89	87	73	58	i= 4
			i pv											j	j=15
10	20	45	74	87	52	46	14	12	51	63	89	87	73	58	i= 4
			i pv											j	j=15
10	20	45	74	87	52	46	14	12	51	63	89	87	73	58	i= 4
			i										p j		j=14
10	20	45	74	87	52	46	14	12	51	63	89	87	73	58	i= 4
			i									p j			j=13
10	20	45	74	87	52	46	14	12	51	63	89	87	73	58	i= 4
			i								p j				j=12
10	20	45	74	87	52	46	14	12	51	63	89	87	73	58	i= 4
			i							p j					j=11
10	20	45	74	87	52	46	14	12	51	63	89	87	73	58	i= 4
			i						pj						j=10
10	20	45	74	87	52	46	14	12	51	63	89	87	73	58	i= 4
			i					pv j							j= 9
10	20	45	12	87	52	46	14	74	51	63	89	87	73	58	i= 4
			i					j	(tauschen)						j= 9
10	20	45	12	87	52	46	14	74	51	63	89	87	73	58	i= 5
				i			j		(i und j versetzen)						j= 8
10	20	45	12	87	52	46	14	74	51	63	89	87	73	58	i= 5
				i pv			j								j= 8
10	20	45	12	87	52	46	14	74	51	63	89	87	73	58	i= 5
				i			pv j								j= 8
10	20	45	12	14	52	46	87	74	51	63	89	87	73	58	i= 5
				i			j		(tauschen)						j= 8
10	20	45	12	14	52	46	87	74	51	63	89	87	73	58	i= 6
					i	j			(i und j versetzen)						j= 7
10	20	45	12	14	52	46	87	74	51	63	89	87	73	58	i= 6
					ipv	j									j= 7
10	20	45	12	14	52	46	87	74	51	63	89	87	73	58	i= 6
					i	pvj									j= 7
10	20	45	12	14	46	52	87	74	51	63	89	87	73	58	i= 6
					i	j			(tauschen)						j= 7
10	20	45	12	14	46	52	87	74	51	63	89	87	73	58	i= 7
					j	i			(i und j versetzen)						j= 6

Ende 1. Durchgang:

— Zeiger auf 1 setzen
— links(1) = links (= 1)
— rechts(1) = j (= 6)
— Zeiger auf 2 setzen
— links(2) = i (= 7)
— rechts(2) = rechts (=15)

Für 2. Durchgang:

— links = links(2)
— i = links
— rechts = rechts(2)
— j = rechts
— Zeiger auf 1 zurücksetzen

2. Durchgang (in etwas verkürzter Form):

```
10   20   45   12   14   46 ! 52   87   74   51   63   89   87   73   58   i= 7
                              i         vergleich = 63                 j   j=15
10   20   45   12   14   46 ! 52   87   74   51   63   89   87   73   58   i= 7
                              i p                                      j   j=15
10   20   45   12   14   46 ! 52   87   74   51   63   89   87   73   58   i= 8
                                   i pv                                j   j=15
10   20   45   12   14   46 ! 52   87   74   51   63   89   87   73   58   i= 8
                                   i                                pv j   j=15
10   20   45   12   14   46 ! 52   58   74   51   63   89   87   73   87   i= 9
(tauschen und versetzen)                i pv                      j        j=14
10   20   45   12   14   46 ! 52   58   74   51   63   89   87   73   87   i= 9
                                        i                       p j        j=14
10   20   45   12   14   46 ! 52   87   74   51   63   89   58   73   87   i= 9
                                        i                  p j             j=13
10   20   45   12   14   46 ! 52   58   74   51   63   89   87   73   87   i= 9
                                        i             p j                  j=12
10   20   45   12   14   46 ! 52   58   74   51   63   89   87   73   87   i= 9
                                        i       pv j                       j=11
10   20   45   12   14   46 ! 52   58   63   51   74   89   87   73   87   i=10
(tauschen und versetzen)                          ij                       j=10
10   20   45   12   14   46 ! 52   58   63   51   74   89   87   73   87   i=10
                                                   ij  (i prüfen)          j=10
10   20   45   12   14   46 ! 52   58   63   51   74   89   87   73   87   i=11
(wird nicht getauscht, da i größer j)        j    i pv                     j=10
```

Ende 2. Durchgang:

- Zeiger auf 2 setzen
- links(2) = links (= 7)
- rechts(2) = j (=10)
- Zeiger auf 3 setzen
- links(3) = i (=11)
- rechts(3) = rechts (=15)

Für 3. Durchgang:

- links = links(3)
- i = links
- rechts = rechts(3)
- j = rechts
- Zeiger auf 2 zurücksetzen

3. Durchgang (verkürzt):

```
10  20  45  12  14  46 ! 52  58  63  51 ! 74  89  87  73  87  i=11
vergleich = 87                            i p                 j  j=15

10  20  45  12  14  46 ! 52  58  63  51 ! 74  89  87  73  84  i=12
                                              i pv            j  j=15

10  20  45  12  14  46 ! 52  58  63  51 ! 74  89  87  73  84  i=12
                                              i            pv j  j=15

10  20  45  12  14  46 ! 52  58  63  51 ! 74  87  87  73  89  i=13
(tauschen und versetzen)                          i p  j         j=14

10  20  45  12  14  46 ! 52  58  63  51 ! 74  87  87  73  84  i=14
                                       (i prüfen)      ij        j=14

10  20  45  12  14  46 ! 52  58  63  51 ! 74  87  87  73  84  i=15
(i prüfen und 'vertauschen', wird aber wegen i         j   i     j=14
größer j nicht durchgeführt)
```

Ende 3. Durchgang:

- Zeiger auf 3 setzen
- links(3) = links (=11)
- rechts(3) = j (=14)
- da i nicht kleiner als rechts ist (beide 15), ist das letzte Element schon 'sortiert' und es wird keine zweite Speicherung in links( ) und rechts( ) vorgenommen.

Für 4. Durchgang:

— links	=	links(3)
— i	=	links
— rechts	=	rechts(3)
— j	=	rechts
— Zeiger auf 2 zurücksetzen		

4. Durchgang (da wir in unserem Beispiel nur 15 Elemente sortieren, wollen wir MINSORT für vier Elemente aufrufen)
10 20 45 12 14 46 ! 52 58 63 51 ! 73 74 87 87 89

Nach dem vierten Durchgang werden keine neuen Teilbereiche festgelegt.

Für 5. Durchgang:

— links	=	links(2)	(= 7)
— i	=	links	
— rechts	=	rechts(2)	(=10)
— j	=	rechts	
— Zeiger auf 1 zurücksetzen			

5. Durchgang (auch hier wieder MINSORT):
10 20 45 12 14 46 ! 51 52 58 63 ! 73 74 87 87 89

Nach dem fünften Durchgang werden keine neuen Teilbereiche festgelegt.

Für 6. Durchgang:

— links	=	links(1)	(=1)
— i	=	links	
— rechts	=	rechts(1)	(=6)
— j	=	rechts	
— Zeiger auf 0 zurücksetzen			

6. Durchgang
(mal wieder QUICKSORT; nur Ergebnis;
Vergleich=12):

```
10   12 ! 45   20   14   46 ! 51   52   58   63   73   74   87   87   89
      j    i   (pv)
```

Die neuen Grenzen sind 1/2 und 3/6. Noch zweimal MINSORT aufrufen, und die Sache ist erledigt:

```
10   12   14   20   45   46   51   52   58   63   73   74   87   87   89
```

Damit wurden alle Spezialfälle behandelt.

Damit die Zahlen alle kleiner als 100 sind, wählen wir als Vergleichskriterium zu Beginn natürlich die Zahl 50. Weil wir aber im allgemeinen nicht wissen, was für Daten sich in dem Feld befinden, eine Überprüfung ist meist nicht möglich, oder kostet zusätzliche Rechenzeit, versuchen wir uns wenigstens bei vorsortierten Daten dem Optimum soweit wie möglich zu nähern. Dazu benutzen wir als Vergleichskriterium ein Element in der Mitte des Feldes, bei einer ungeraden Anzahl von Elementen, das Mittlere. Die jeweils rechte und linke Grenze der sortierten Teilbereiche müssen wir uns natürlich merken, was wir später in Basic in zwei Feldern RE( ) und LI( ) tun werden. Ein Zeiger (ZE) zeigt dabei auf die nächsten zu bearbeitenden Elemente aus RE( ) und LI( ).

Hier nun die Dokumentation für QUICK-SORT nach Zahlen:

Name: QUICKZAHL

Zweck: QUICKSORT für Zahlen

```
1000 REM ------------------------------
1010 REM - TESTRAHMEN FUER QUICKSORT --
1020 REM ------------------------------
1030 :
1060 AN=500
1070 DIM WE(ANZAHL)
1071 DIM LI(AN/2)
1072 DIM RE(AN/2)
1080 :
1090 FOR I=1 TO AN
1100 : WE(I)=INT(RND(TI)*10*AN)+1
1110 NEXT
1120 :
1130 T=TI
1140 GOSUB 9000
1150 T=(TI-T)/60
1160 :
1170 FOR I=1 TO AN
1180 : PRINT I,WE(I)
1190 NEXT
1200 :
1210 PRINT"BENOETIGTE ZEIT ";T;"SEKUNDEN = ";T/60;"MINUTEN"
1220 :
1230 END
1240 :
8000 REM ------------------------------
8010 REM ---         MINSORT        ---
8020 REM ------------------------------
8030 :
8040 PRINT "MINSORT ",LI,RE
8050 FOR K=LI TO RE-1
8060 : MI=1000000
8070 :
8080 : FOR L=K TO RE
8090 : : IF WE(L) < MI THEN MI=WE(L) : WO=L
8100 : NEXT
8110 :
8120 : HI=WE(K) : WE(K)=WE(WO) : WE(WO)=HI
8130 NEXT
8140 :
8150 RETURN
8160 :
9000 REM ------------------------------
9010 REM ---        QUICKSORT       ---
9020 REM ------------------------------
9030 :
9050 ZE   = 1
9060 LI(1)= 1
9070 RE(1)= AN
9080 LI   = LI(ZE)
9090 RE   = RE(ZE)
9100 ZE   = ZE-1
9110 :
```

Listing 4/6.2.1-5 (Teil 1)

```
9120 IF (RE-LI)<15 THEN GOSUB 8000 : GOTO 9400
9130 VE = WE((LI+RE)/2)
9140 I=LI
9150 J=RE
9160 :
9170 PRINT "QUICKSORT LINKS";I,"RECHTS";J,"VERGLEICH";VE
9180 :
9190 FOR Z=1 TO 1000000
9200 : IF WE(I) <= VE AND I<J THEN I=I+1 : GOTO 9200
9210 : IF WE(J)  > VE AND J>I THEN J=J-1 : GOTO 9210
9220 : IF I=>J THEN 9340
9230 :
9240 : HI    = WE(I)
9250 : WE(I) = WE(J)
9260 : WE(J) = HI
9270 :
9280 : I=I+1
9290 : J=J-1
9300 :
9310 : IF J <= I THEN 9340
9320 NEXT
9330 :
9340 IF I=J AND I=RE THEN J=J-1
9350 IF I=J AND J=LI THEN I=I+1
9360 IF LI<J THEN ZE=ZE+1 : LI(ZE)=LI : RE(ZE)=J
9370 IF I<RE THEN ZE=ZE+1 : LI(ZE)=I : RE(ZE)=RE
9380 :
9390 FOR X=1 TO ZE : PRINT "ZEIGER";X;":";LI(X),RE(X) : NEXT
9400 IF ZE>0 THEN 9080
9410 :
9420 RETURN
```

Listing 4/6.2.1-5 (Teil 2)

Struktogramm:

- Wert für Zeiger auf Grenzfelder initialisieren (=1)
- Grenzen des zu sortierenden Feldes in die Grenzfelder (LI( ) und RE( ) bei Index 1 eintragen
- solange Teilbereiche zu sortieren sind
  - Grenzen für den nächsten Sortiervorgang (LI und RE aus den Grenzfeldern übernehmen)
  - Zeiger in Grenzfelder eine Einheit zurücksetzen
  - Wenn weniger als 15 Elemente zu sortieren sind, dann MINSORT aufrufen
  - Vergleichselement festlegen (siehe Text)
  - 'Laufvariablen' I und J auf die äußeren Grenzen setzen
  - solange der Sortiervorgang nicht beendet ist
    - solange das aktuelle Element, auf das die Laufvariable I zeigt, kleiner oder gleich dem Vergleichselement ist und I noch nicht den Wert von J erreicht hat
      - erhöhe Wert von I um 1 (aktuelle linke Grenze nach rechts)
    - solange das aktuelle Element, auf das die Laufvariable J zeigt, größer als das Vergleichselement ist und J größer I ist
      - vermindere Wert von J um 1 (aktuelle rechte Grenze nach links)
    - Ist I größer oder gleich J?
      - nein:
        - Vertausche Elemente mit Indices I und J
        - erhöhe I um i
        - vermindere j um 1
      - ja:
        - Ist ursprüngliche linke Grenze links von J?
          - ja:
            - erhöhe Zeiger auf Grenzfelder um 1
            - merke ursprüngliche linke Grenze im Grenzfeld für links
            - merke J im Grenzfeld für rechts
          - nein:
            - Ist ursprüngliche rechte Grenze rechts von I?
              - ja:
                - erhöhe Zeiger auf Grenzfelder um 1
                - merke I im Grenzfeld für links
                - merke ursprünglich rechte Grenze im Grenzfeld für rechts
              - nein:
- Rücksprung

Variablen:

Name	zul. Bereich	Bedeutung
Eingabeparameter:		
WE( )	Dezimalzahlen	zu sortierende Daten
Ausgabeparameter:		
WE( )	Dezimalzahlen	sortierte Daten
Lokale Variablen: (alle Integer kleiner als AN)		
HI	Dezimalzahl	zum Vertauschen
I	Integer	linke Grenze des aktuellen Durchlaufes
J	Integer	rechte Grenze des aktuellen Durchlaufes
LI	Integer	linke Grenze des aktuellen Teilbereiches
LI( )	Integer	linke Grenzen der noch zu sortierenden Teilbereiche
RE	Integer	rechte Grenze des aktuellen Teilbereiches
RE( )	Integer	rechte Grenzen der noch zu sortierenden Teilbereiche
VE	Dezimalzahl	Vergleichselement (s. Text)
ZE	Integer(AN/2)	Zeiger in die Hilfsfelder der zu sortierenden Teilbereiche
Globale Variablen:		
AN	Integer	Größe des zu sortierenden Feldes

Weitere Unterprogrammaufrufe:

in Zeile	nach Zeile	Zweck	
9120	8100	MINZAHL	muß angepaßt werden

Rücksprünge:

In Zeile	Art	Bedingung	Bemerkung
9420	Return	—	fertig

Wegen der Komplexität des Algorithmus wollen wir jedoch noch auf einige wesentliche Punkte eingehen. Außer dem Feld für die zu sortierenden Daten müssen noch zwei Felder LI( ) und RE( ) im Hauptprogramm dimensioniert werden. Die Dimensionierung braucht maximal nur bis zur Hälfte der Anzahl der zu sortierenden Datenelemente zu gehen, da wir ja nicht bis zu einem einzelnen Element herunter mit QUICKSORT arbeiten, meist reichen auch kürzere Felder. In die ersten Elemente von LI( ) und RE( ) tragen wir die äußeren Grenzen aller zu sortierenden Daten ein, also 1 und AN. Auch den Zeiger auf den nächsten zu sortierenden Teilbereich setzen wir auf 1, da wir schließlich zu Beginn das ganze Feld bearbeiten müssen.

Die Zeilen 9050 bis 9070 werden nur einmal benötigt. Bitte beachten Sie, daß Zeile 9080 bereits zur Programmschleife gehört, in der QUICKSORT sich immer wieder selbst aufruft. Daher können wir auch eine Zuweisung in der Form 'LI=1' und 'RE=AN' nicht durchführen, sondern verwenden gleich die entsprechenden Feldelemente. Wenn wir einen neuen Durchlauf starten, müssen wir natürlich auch den Zeiger auf die aktuellen Grenzen vermindern (9100).

Die Variablen RE und LI beinhalten also Indices auf das zu sortierende Feld. Wenn ihre Differenz kleiner als 15 ist, haben wir es nur mit einem kleinen Teilbereich zu tun, den wir aus den genannten Gründen nicht mit QUICKSORT, sondern MINSORT sortieren wollen.

Wenn wir das Verfahren MINSORT angewendet haben (das noch auf unsere speziellen Belange angepaßt werden muß, wie aus den Zeilen 8050, 8080, 8090 und 8120 hervorgeht), können wir gleich zur Überprüfung übergehen, ob überhaupt noch ein Teilbereich zu sortieren ist. Dies ist der Fall, wenn unser Hilfselement ZE noch nicht den Wert 0 erreicht hat. Die Variable ZE gibt also nicht nur den aktuellen Index von LI( ) und RE( ) an, sondern zählt auch noch die zu sortierenden Teilbereiche.

Hat der aktuell zu sortierende Teilbereich 15 oder mehr Elemente, so wird er mit QUICKSORT sortiert. Dazu wird in Zeile 9130 das Vergleichselement nach der bereits erwähnten Methode ermittelt. Außerdem werden die beiden Laufvariablen auf die äußeren Grenzen des zu sortierenden Teilbereichs gesetzt. Zeile 9170 soll nur andeuten, daß QUICKSORT seine Arbeit aufgenommen hat, und anhand der Grenzen kann man den Sortierfortschritt erkennen, gegebenenfalls können sie die PRINT-Befehle löschen oder mit REM ausklammern.

Die Zeilen 9150 bis 9320 bilden eine prinzipielle Endlosschleife. Die Laufvariable I läuft nun vom linken Rand zur Mitte und Laufvariable J vom rechten Rand zur Mitte. Abbruchkriterium ist in den Zeilen 9220/9310 der Vergleich von I und J. Wenn I größer als J ist, ist der Teilbereich zu Ende sortiert, und die Werte werden als Rand für die beiden sortierten Teilbereiche herangezogen.

Zeile 9240 wird erreicht, wenn der Teilbereich noch nicht zu Ende sortiert wurde. Für den nächsten Suchvorgang müssen natürlich die aktuellen Laufvariablen entsprechend erhöht bzw. vermindert werden.

Nachdem der Teilbereich sortiert wurde, ergeben sich zwei neue Teilbereiche, einer vom linken Rand bis zu der durch J markierten Stelle, und einer vom rechten Rand bis zu der von I markierten Position. Beide Grenzen werden in die Felder LI( ) und RE( ) eingetragen, wozu auch noch der Zeiger auf das aktuelle Feldelement erhöht wird. Es sei noch darauf hingewiesen, daß die Variable ZE nicht erhöht wird, wenn das Unterprogramm MINSORT aufgerufen wird.

Die Zeilen 9340 und 9350 behandeln einen Spezialfall, nämlich wenn I und J gleich sind, und außerdem noch auf einer der beiden äußeren Begrenzungen. Hier würde der 'liebste Fehler' eines Programmierers auftreten: eine Endlosschleife. Der aktuelle Teilbereich würde immer wieder neu sortiert. Wir schaffen Abhilfe, indem wir die rechte oder linke Grenze — je nachdem, wo sich I und J befinden — um eine Einheit zur Mitte schieben. Der Fall tritt bei unglücklicher Wahl des Vergleichselementes auf.

Sie sehen, daß die Verwaltung der einzelnen Teilbereiche bei QUICKSORT sehr aufwendig ist. Versuche haben ergeben, daß andere Sortierverfahren bei 10 bis 15 zu sortierenden Elementen schneller sind, so daß QUICK-SORT immer in Verbindung mit einem anderen Sortierverfahren verwendet werden sollte. Wie der Sortiervorgang im Hinblick auf die Teilbereiche und Zeiger abläuft, zeigt Ihnen Bild 4/6.2.1-6.

```
QUICKSORT LINKS 1               RECHTS 100           VERGLEICH 787
ZEIGER 1 : 1               80
ZEIGER 2 : 81               100
QUICKSORT LINKS 81               RECHTS 100           VERGLEICH 809
ZEIGER 1 : 1               80
ZEIGER 2 : 81               84
ZEIGER 3 : 85               100
QUICKSORT LINKS 85               RECHTS 100           VERGLEICH 967
ZEIGER 1 : 1               80
ZEIGER 2 : 81               84
ZEIGER 3 : 85               97
ZEIGER 4 : 97               100
MINSORT               97               100
MINSORT               85               97
MINSORT               81               84
QUICKSORT LINKS 1               RECHTS 80           VERGLEICH 522
ZEIGER 1 : 1               54
ZEIGER 2 : 55               80
```

Bild 4/6.2.1-6 (Teil 1)

```
QUICKSORT LINKS 55              RECHTS 80            VERGLEICH 576
ZEIGER 1 : 1             54
ZEIGER 2 : 55             64
ZEIGER 3 : 65             80
QUICKSORT LINKS 65              RECHTS 80            VERGLEICH 588
ZEIGER 1 : 1             54
ZEIGER 2 : 55             64
ZEIGER 3 : 65             67
ZEIGER 4 : 68             80
MINSORT              68              80
MINSORT              65              67
MINSORT              55              64
QUICKSORT LINKS 1              RECHTS 54            VERGLEICH 150
ZEIGER 1 : 1             12
ZEIGER 2 : 13             54
QUICKSORT LINKS 13              RECHTS 54            VERGLEICH 373
ZEIGER 1 : 1             12
ZEIGER 2 : 13             44
ZEIGER 3 : 45             54
MINSORT              45              54
QUICKSORT LINKS 13              RECHTS 44            VERGLEICH 278
ZEIGER 1 : 1             12
ZEIGER 2 : 13             29
ZEIGER 3 : 30             44
MINSORT              30              44
QUICKSORT LINKS 13              RECHTS 29            VERGLEICH 222
ZEIGER 1 : 1             12
ZEIGER 2 : 13             20
ZEIGER 3 : 21             29
MINSORT              21              29
MINSORT              13              20
MINSORT              1              12
```

Bild 4/6.2.1-6 (Teil 2)

Es ist offensichtlich, daß die benötigte Rechenzeit mit der Anzahl der zu sortierenden Elemente anwächst. Allerdings ist — wie Sie am Beispiel QUICKSORT sehen — nicht nur die Anzahl der Elemente ausschlaggebend. Wenn man nur die Anzahl der Elemente betrachtet (n), so ist bei SHELL-SORT der Aufwand proportional n^2 bei MINSORT/MAXSORT $n^2/2$ und bei QUICK-SORT $n*\ln(n)$. Nimmt man den Programmaufwand (Prüfen und gegebenenfalls Vertauschen), so ergibt sich nach Knuth (Arts of Computerprogramming, Art 3, Sorting and Searching) für ein primitives Verfahren wie unser erstes Beispiel ein Aufwand von $2*n^2+9*n$ und für QUICK-SORT ein Aufwand von $12*n*\ln(n)+2*n$.

Zwei Faktoren sind bei QUICKSORT jedoch von entscheidender Bedeutung, wenn man von dem Programmaufwand und der Anzahl zu sortierender Elemente ein-

mal absieht: Das Vergleichselement und die verwendete Programmiersprache bzw., der verwendete Rechner. Da der Programmaufwand bei QUICKSORT sehr groß ist, ergibt sich bei kompilierten Programmiersprachen ein deutlich größerer Vorteil. Ein ungeschickt gewähltes Vergleichselement kann andererseits den ganzen Rechenzeitvorteil von QUICKSORT zunichte machen. Optimum ist, wenn jeder zu sortierende Teilbereich halbiert wird.

Bevor wir kurz auf einige andere Sortierverfahren eingehen, noch eine kleine Tabelle mit den Rechenzeiten der drei vorgestellten Algorithmen:

	Sortierte Daten			Unsortierte Daten		
AN	SHELL	MIN/MAX	QUICK	SHELL	MIN/MAX	QUICK
10	0,2 s	0,5 s	1 s	2 s	0,5 s	1 s
50	0,8 s	8 s	11 s	48 s	8 s	12 s
100	2 s	28 s	25 s	191 s	28 s	29 s
500	8 s	11 s	168 s	40 m	11 m	220 s
1000	15,5 s	42 m	6 m	5,5 h	42 m	7,5 m

(circa-Angaben in: s=Sekunden; m=Minuten; h=Stunden)
(QUICKSORT mit MINSORT bei weniger als 15 Elementen)

Andere Sortierverfahren

Zum Abschluß möchten wir noch kurz auf zwei andere Sortierverfahren eingehen. Wir haben zur Darstellung des Ablaufes der genannten Sortierverfahren immer das Beispiel ‘Sortieren von Karteikarten’ verwendet. Im normalen Büroalltag wird man sicherlich so vorgehen, daß man bei großen Mengen zunächst nach den Anfangsbuchstaben A bis Z 26 Stapel bildet, und diese dann in sich sortiert. Sind diese 26 Stapel auch wieder sehr groß, wird man dann nach dem zweiten Buchstaben vor sortieren. Diese Vorgehensweise ist auch auf den Computer übertragbar, und das zugehörige Sortierprogramm wird RADIXSORT genannt.

Beim Computer, insbesondere der Datenverarbeitung im kommerziellen Bereich, verwendet man für große Datenmengen (Kundenstammdaten etc.) bei dem Zugriffsschlüssel auf einen Datensatz sehr häufig Suchbäume. Der einfachste Suchbaum ist der Binär-Baum (vergleiche Kapitel 4/4.7). Abgesehen davon, daß Suchbäume einen schnellen, gezielten Zugriff auf bestimmte Datensätze ermöglichen, läßt sich durch einen geeigneten Weg in einem Suchbaum auch eine sortierte Reihenfolge erzielen. Liegt also ein solcher Suchbaum, z.B. ein Binär-Baum, vor, so kann das Sortieren entfallen, da nur noch der Suchbaum abgearbeitet werden muß.

Es gibt auch ein Sortierverfahren HEAP-SORT, das zunächst einen Binär-Baum aufgrund der Daten aufstellt und diesen dann in der sortierten Reihenfolge abarbeitet. In Basic ist das Aufstellen eines Binär-Baumes ein ziemlicher Rechenaufwand, so daß das Verfahren HEAP-SORT für eine Sortier-Anwendung meist nicht geeignet erscheint. Auch ist die Komplexität des Vorganges nicht so leicht zu durchschauen, wie bei den vorgestellten Verfahren. Bei Programmiersprachen, die Variablen in sogenannter Verbundstruktur zulassen, wie z.B. PASCAL, und die außerdem durch eine Compilierversion relativ schnell sind, wird das Verfahren über Suchbäume häufig eingesetzt.

4/6.3

Assembler-Utilities

Bei den Utilities sollen neben Basic-Programmen und allgemeinen Routinen natürlich auch Assembler-Utilities nicht zu kurz kommen. Zu unterscheiden ist hier zwischen eigenständigen Assembler-Utilities und solchen, die als Basic-Erweiterung eingebunden werden. Bei vielverwendeten Utilities ist es sicherlich besser, diese über einen neuen Basic-Befehl, gegebenenfalls mit Parametern, aufzurufen. Diesem Bereich sind die Kapitel 4/6.4 für Erweiterung beim C 64 und 4/6.5 für Erweiterung beim C 128 vorbehalten.

Zunächst sind als Assembler-Utilities ein Interrupt-Manager und Zahlkonvertierungen vorgesehen. Der Interruptmanager erlaubt es Ihnen, mehrere Programme gleichzeitig (jedenfalls aus Sicht des Anwenders) ablaufen zu lassen, wobei die Interrupt-Routine der Commodore-Rechner benutzt wird.

Zahlkonvertierungen unterstützen allgemein die Arbeit mit dem Rechner, da man bei intensiver Arbeit ständig mit Dezimalzahlen, Binärzahlen und Hexadezimalzahlen arbeiten muß.

4/6.3.1

Interrupt-Manager

(Autor: Werner Eberl)

Der C 128 weist gegenüber dem C 64 einige funktionelle Merkmale auf, die dem Benutzer das Programmieren dieses Rechners wesentlich erleichtern, so z.B. viele praktische Basic-Befehle und ein besser stukturiertes und umfangreicheres Betriebssystem.

Was aber immer noch fehlt, ist eine Möglichkeit, mehrere Programme gleichzeitig nebeneinander laufen zu lassen, etwa im Sinne eines Multi-Tasking-Betriebes. Um dies wenigstens für Maschinenprogramme zu ermöglichen, stellen wir Ihnen im folgenden einen Interrupt-Manager vor, der es ermöglicht, bis zu acht kleine Programme quasi gleichzeitig ablaufen zu lassen. Bei jedem Interrupt (also alle 1/60 Sekunden) werden diese Programme nacheinander abgearbeitet.

Mögliche Anwendungen sind:

- Keyclick; bei jedem Tastendruck ertönt ein Klick im Lautsprecher.
- Spezielle Musik- oder Grafikroutinen; dynamische Vorgänge wie etwa das Auf- und Abschwellen eines Tones oder dynamische Grafikeffekte lassen sich leicht in einem kleinen Maschinenprogramm formulieren, das dann vom Interrupt-Manager aufgerufen wird.
- Blinkeffekte mit Rahmen- oder Hintergrundgarbe
- Anschluß zusätzlicher Tasten über Joystick-Anschluß
- Bedienung selbstgebauter Hardware-Erweiterungen
- und vieles mehr

Wir wollen an dieser Stelle keine der oben aufgeführten Programme abbilden, sondern nur die notwendigen Hilfszellen und das eigentliche Programm des Interrupt-Managers darstellen. Ebenso nicht abgebildet ist die Routine zum Umbiegen des IRQ-Vektors auf unsere Manager-Routine. Dieses kleine Problem können Sie jedoch in ein paar Maschinenprogrammzeilen lösen, wobei Sie beachten müssen, daß Sie während des Umstellens des Vektors das Interrupt-Disable-Flag setzen müssen. Die nötigen Befehle sind z.B. in der Initialisierungs-Routine für unsere Basic-Erweiterungen enthalten.

```
1303 4C 28 13            JMP INTMAN1
1306 00        IRQM:     .BY 0          ;MASKE F. IRQ-ROUT.
               ISPTAB:                  ;SPRUNGTAB. IRQ-ROUT.
1307 1A 13               .WO IDUMMY     ;NICHTS-TU-ROUTINE
1309 1A 13               .WO IDUMMY
130B 1A 13               .WO IDUMMY
130D 1A 13               .WO IDUMMY
130F 1A 13               .WO IDUMMY
1311 1A 13               .WO IDUMMY
1313 1A 13               .WO IDUMMY
1315 1A 13               .WO IDUMMY
1317 1A 13     IADR:     .WO IDUMMY     ;PLATZ F. SPRUNGZ.
1319 00        IRQZ:     .BY 0          ;ZAEHLER IRQ-ROUTINEN
131A 60        IDUMMY:   RTS            ;DUMMY-ROUTINE
               ;
```

Listing 4/6.3.1-1

Die erste Hilfszelle mit dem Namen IRQM stellt das Maskenregister für die acht Routinen dar. Ist das entsprechende Bit des Registers gleich 1, wird die zugehörige Routine bei jedem IRQ-Interrupt aufgerufen, sonst nicht.

Als nächstes folgt die Sprungtabelle mit den Startadressen der einzelnen Interrupt-Routinen. In der vorliegenden Version des Interrupt-Managers ist es nur möglich, Routinen in den ersten 16 K der Bank 0 unterzubringen. Für die meisten Anwendungen dürfte dies jedoch ausreichen, da selbst mit dem Basic-Befehlsdekoder und dem Basic-Funktionsdekoder aus Kapitel 4/6.5.1 noch etwa 1 KByte frei ist. Die Zieladressen müssen in der Form Low Byte, High Byte eingetragen werden.

Im abgedruckten Programmtext steht an Stelle der eigentlichen Ziele die Routine IDUMMY, die schlicht einen RTS-Befehl enthält, also nichts tut. Dies ist notwendig, damit der Rechner nicht ins Leere springt, falls versehentlich einmal ein Bit des Maskenregisters IRQM gesetzt wurde. An dieser Stelle sollte gesagt werden, daß die einzelnen Routinen immer mit RTS aufhören müssen, weil sie vom Interrupt-Manager im Prinzip mit einem JSR-Befehl aufgerufen werden. Die Verwendung des RTS-Befehls als Endemarke hat auch den Vorteil, daß beim Testen die Routine einfach mit SYS aufgerufen werden kann.

Die Zelle IADR dient zur Aufnahme des späteren Sprungziels, die Zelle IRQZ markiert die aktuell angesprochene Routine, indem an der entsprechenden Stelle eine 1 gesetzt wird.

```
                 EIRQ:                          ; ** INTERRUPT- **
                 ;              ; ** MANAGER    **
1318 A9 01                  LDA #1
131D 8D 19 13               STA IRQZ            ;ZAEHLER AUF 1
                 INTMANS:
1320 AD 19 13               LDA IRQZ            ;ZAEHLER UND-VERKN.
1323 2D 06 13               AND IRQM            ;MIT MASKE
1326 F0 03                  BEQ INTMAN1         ;FLAG NICHT GES.
1328 20 33 13               JSR INTAUSF         ;EINEN INT. AUSF.
                 INTMAN1:
132B 0E 19 13               ASL IRQZ            ;ZAEHLERBIT ROTIEREN
132E 90 F0                  BCC INTMANS         ;SCHLEIFE
1330 4C 65 FA               JMP NIRQ            ;FERTIG
                 ;
                 INTAUSF:
1333 A2 FE                  LDX #$FE            ;X = -2
                 INTAUSF1:
1335 E8                     INX                 ;X UM 2 ERHOEHEN
1336 E8                     INX
1337 4A                     LSR                 ;AKKU NACH RECHTS
1338 90 FB                  BCC INTAUSF1        ;BIS NUMMER GEFUNDEN
133A BD 08 13               LDA ISPTAB+1,X      ;SPRUNGZIEL HIGH/LOW
133D 8D 18 13               STA IADR+1          ;IN RESERVIERTE
1340 BD 07 13               LDA ISPTAB,X        ;ZELLEN SCHREIBEN
1343 8D 17 13               STA IADR
1346 6C 17 13               JMP (IADR)          ;IND. SPRUNG BANK 0
                 ;
                 ;
```

Listing 4/6.3.1-2

Das eigentliche Interrupt-Manager-Programm beginnt mit dem Label EIRQ. Das ist der Wert, auf den der Vektor IIRQ ($0314, $0315) gesetzt werden muß. Der Standardwert dieses Vektors ist das Label NIRQ (=$FA65).

Der Manager beginnt damit, eine 1 in die hinterste Stelle des Zählerbytes IRQZ zu schreiben. Die Schleife für alle acht Routinen befindet sich zwischen den Labels INTMANS und den darauffolgenden BCC-Befehl. Dabei wird jeweils das Zählerregister mit dem Maskenregister UND-verknüpft und, wenn in beiden Zellen an der entsprechenden Stelle eine 1 war, zur Interrupt-Ausführroutine gesprungen, sonst nur das Zählerbit weitergeschoben und zur Schleife gesprungen. Die Schleife wird beendet, wenn das Zählerbit in das Carry-Flag geschoben wurde. Daraufhin wird zur Standard-Routine NIRQ gesprungen.

Die Interrupt-Ausführroutine wandelt zunächst die Information des Zählerbits (die Stelle, wo die 1 steht) in einen Offset-Zeiger für die Interrupt-Sprungtabelle um. Die entsprechenden Sprungzielwerte werden in die Zellen IADR und IADR+1 kopiert und dann ein indirekter Sprung ausgeführt. Um Sprünge in andere RAM-Bänke durchzuführen, müßte man anstatt des indirekten Sprungs eine Routine anspringen, die ähnlich der bei den Basic-Erweiterungen aufgeführten Weitsprungsequenz aufgebaut ist (vgl. Kapitel 4/6.5.1.2).

Um Ihnen die Vorgehensweise und die Möglichkeiten des Interrupt-Managers vor Augen zu führen, bieten wir nun ein ganz kleines Beispielprogramm an, daß einfach ca. alle vier Sekunden die Rahmenfarbe ändert. Wollen Sie diese Routine in den Manager einbauen, so müssen Sie die Adresse dieser Routine in eine der Zellen der Interrupt-Sprungtabelle eintragen und dann das entsprechende Bit im Maskenregister IRQM setzen.

```
14FF    00          ZAEHL:    .BY    0
1500    CE FF 14    TESTI:    DEC    ZAEHL
1503    D0 03                 BNE    FERTIG
1505    EE 20 D0              INC    $D020; Rahmenfarbe
1508    60          FERTIG:   RTS
```

Zum Testen können Sie folgendes Programm verwenden:

```
10 FOR   I=1 TO 10
20 READ  A$
30 POKE  DEC ("14FF")+I—1,DEC(A$)
40 NEXT
50 DATA  0,CE,FF,14,D0,03,EE,20,D0,60
60 POKE  DEC("1307"),DEC("00")     : REM Low-Adresse
70 POKE  DEC("1308"),DEC("15")     : REM High-Adresse
80 POKE  DEC("1306"),1             : REM Maskenregister
```

Hier wird also nichts anderes durchgeführt als fortlaufend ein Zähler vermindert und, wenn dieser null ist, die Rahmenfarbe erhöht.

4/6.3.2

Zahlenkonvertierung

(Autor: Werner Eberl)

Häufig benötigte Unterprogramme sind diejenigen zur Umwandlung von Zahlenformaten. Die wichtigsten Zahlenformate sind:

— Fließkommazahlen
— Ganze Zahlen in 16-Bit-Darstellung
— Zeichenreihen.

Die folgenden Routinen sind so zusammengestellt, daß einheitliche Konventionen über die Parameterübergabe eingehalten werden können. Im einzelnen wurde folgendes vereinbart:

— Fließkommazahlen werden im Fließkomma-Akkumulator als Ein- oder Ausgabeparameter übergeben.

— Positive ganze Zahlen von 0 bis 65535, auf englisch heißen solche Zahlen non signed integers. Sie werden in den Prozessor-Registern Akku und Y übergeben, wobei das höherwertige Byte im Akku steht.

— Ganze Zahlen im Bereich von -32768 bis +32767, sogenannte vorzeichenbehaftete ganze Zahlen, werden in Zweierkomplementdarstellung im Akku und Y-Register übergeben, wobei wieder das höherwertige Byte, das hier auch das Vorzeichen enthält, in den Akku gehört.

— Zeichenreihen werden — unabhängig davon, ob sie in hexadezimaler, binärer oder dezimaler Form vorliegen — ab dem Label PUFFER-1 (=$00FF/255) abgelegt. Das Ende der Zeichenreihe wird mit einem 0-Byte markiert. Das erste Zeichen der Reihe ist bei dezimaler Darstellung ein Leerzeichen, bei hexadezimaler Darstellung ein Dollarzeichen und bei binärer Darstellung ein Prozentzeichen.

Integer nach Fließkomma

Eine vorzeichenlose ganze Zahl in 16-Bit-Darstellung kann mit der ROM-Routine NOSFLT, die bei $84C9 beginnt, in eine Fließkommazahl, die im FAC gespeichert wird, umgewandelt werden.

Fließkomma nach Integer

Für die Umkehrung des oben beschriebenen Vorgangs gibt es leider keine ROM-Routine, die unseren Parameterkonditionen genügt, so daß wir hier ein Programm

mit 21 Byte angeben müssen, was diesen Zweck erfüllt. Zu beachten ist, daß im Gegensatz zu der Routine GETADR des ROMs diese Routine nicht den Inhalt der Zellen POKER, POKER+1 verändert. Die Routine verzweigt zu einer Fehlermeldung, wenn das Vorzeichen negativ ist oder Exponent größer als 64 ist. Die eigentliche Umwandlung geschieht mit der ROM-Routine QINT, die das Ergebnis im niederwertigsten Byte des FAC hinterlegt, aus welchem wir sie lesen können.

```
                   FLPNOS:
045D A5 68                      LDA FACSGN        ;VORZEICHEN TESTEN
045F 30 0E                      BMI ILLQ          ;NEGATIV, DANN FEHLER
0461 A5 63                      LDA FACEXP        ;EXPONENTEN TESTEN
0463 C9 91                      CMP #145          ;GROESSER ALS 64
0465 B0 08                      BCS ILLQ          ;DANN FEHLER
0467 20 C7 8C                   JSR QINT          ;INTEGER BILDEN
046A A5 66                      LDA FACEXP+3      ;HIGH-BYTE
046C A4 67                      LDY FACEXP+4      ;LOW-BYTE
046E 60                         RTS
                   ;
046F 4C 28 7D      ILLQ:        JMP FCERR         ;'ILL. QANT. ERR.'
                   ;
```

Listing 4/6.3.2-1

Signed-Integer nach Fließkomma

Hier haben wir es wieder einfach, weil hier im ROM eine entsprechende Routine vorhanden ist. Sie hat allerdings einen seltsamen Namen: GIVAYF und beginnt bei $793C.

Fließkomma nach Signed-Integer

Auch hier geht's einfach und zwar mit der Routine FLPINT ab $84B4.

Dezimalstring nach Fließkomma

Diese Aufgabe entspricht der Basic-Funktion VAL. Wir können auch die entsprechende Basic-Befehls-Routine ab $804A verwenden, wobei wir jedoch beachten müssen, daß diese Routine erwartet, daß ein Stringdeskriptor auf dem Stringstapel liegt, wie dies z.B. der Fall ist, wenn ein Stringparameter mit der Routine FRMEVL geholt wird.

Die andere Möglichkeit ist, das Label VAL1, das bei $8052 liegt, zu benützen. Dann müssen wir jedoch die Startadresse des Strings in die Zellen INDEX, INDEX+1 ($24,$25) und die Länge im Akku übergeben. Außerdem muß sichergestellt sein, daß der String eine nichtverschwindende Länge besitzt.

Fließkomma nach Dezimalstring

Diese Umwandlung entspricht der STR$-Funktion. Die entsprechende Standard-Routine aus dem ROM macht zwar das, was wir wollen, aber sie macht auch zuviel:

Sie kopiert den String aus dem Pufferbereich in die Bank 1 und legt einen Stringdeskriptor des neuen Strings auf den Springstapel. Das ist in manchen Fällen sogar praktisch; wenn er aber stört, muß man ihn mit einem Aufruf von FRESTR wieder entfernen.

Hexadezimal nach Integer

Handwerk hat goldenen Boden und deshalb werden wir hier die gesamte Umwandlungsroutine programmieren, ohne auf das ROM zurückzugreifen. Anstelle des Akkus und des Y-Registers wollen wir jedoch die Hilfszellen H2, H3 (=$64,$65) benutzen. Im Y-Register übergeben wir dann der Routine die maximale Anzahl von hexadezimalen Ziffern, die sie auswerten soll. Nach der Umwandlung gibt uns die Routine in diesem Register die Zahl der tatsächlich umgewandelten Ziffern zurück.

```
                  HEXINT:
0472 84 63                   STY H1          ;MAXIMALZAHL MERKEN
0474 A0 00                   LDY #0
0476 84 64                   STY H2
0478 84 65                   STY H3
                  HEXINT1:
047A C4 63                   CPY H1          ;MAXIMALZAHL VERGL.
047C F0 2B                   BEQ HEXINT9
047E B9 00 01                LDA PUFFER,Y    ;NAECHSTES ZEICHEN
0481 38                      SEC
0482 E9 30                   SBC #$30        ;CODE FUER "0" SUB.
0484 C9 0A                   CMP #10
0486 90 0A                   BCC HEXINT2     ;KLEINER 10, DANN OK
0488 E9 07                   SBC #7          ;"A" ENTSPRICHT 10
048A C9 0A                   CMP #10
048C 90 1B                   BCC HEXINT9     ;KLEINER 10, DANN E.
048E C9 10                   CMP #16
0490 B0 17                   BCS HEXINT9     ;>=16, DANN ENDE
                  HEXINT2:
0492 06 65                   ASL H3          ;H3,H2 MIT 16
0494 26 64                   ROL H2          ;MULTIPLIZIEREN
0496 06 65                   ASL H3
0498 26 64                   ROL H2
049A 06 65                   ASL H3
049C 26 64                   ROL H2
049E 06 65                   ASL H3
04A0 26 64                   ROL H2
04A2 05 65                   ORA H3          ;WERT IN DIE LETZTEN
04A4 85 65                   STA H3          ;4 BIT EINTRAGEN
04A6 C8                      INY             ;NAECHSTE HEXZIFFER
04A7 D0 D1                   BNE HEXINT1
                  HEXINT9:
04A9 60                      RTS
                  ;
```

Integer nach Hexadezimal

Auch hier wollen wir eine ausprogrammierte Routine vorstellen, obwohl man in vielen Fällen die HEX$-Routine des ROM benützen kann.

Im Y-Register wird der Routine übergeben, aus wievielen hexadezimalen Ziffern die Zahl bestehen soll. Auch hier werden die Zellen H2, H3 an Stelle von Akku und Y-Register als Eingabeparameter verwendet. Die Zeichenreihe wird ab $1010=256 abgelegt.

```
                  INTHEX:
0598 A9 00                 LDA #0
059A 99 00 01              STA PUFFER,Y      ;ENDEMARKIERUNG
059D 88                    DEY
                  INTHEX1:
059E A5 65                 LDA H3
05A0 29 0F                 AND #$0F          ;4 BIT VON H3
05A2 C9 0A                 CMP #$0A
05A4 90 02                 BCC INTHEX2       ;KLEINER 10
05A6 69 06                 ADC #$06          ;CARRY GESETZT !
                  INTHEX2:
05A8 69 30                 ADC #$30          ;CARRY CLEAR !
05AA 99 00 01              STA PUFFER,Y      ;EINTRAGEN
05AD 46 64                 LSR H2            ;H3,H2 DURCH
05AF 66 65                 ROR H3            ;16 TEILEN
05B1 46 64                 LSR H2
05B3 66 65                 ROR H3
05B5 46 64                 LSR H2
05B7 66 65                 ROR H3
05B9 46 64                 LSR H2
05BB 66 65                 ROR H3
05BD 88                    DEY
05BE 10 DE                 BPL INTHEX1       ;SCHLEIFE
05C0 60                    RTS
                  ;
```

Listing 4/6.3.2-3

Binär nach Integer

Die Vorgehensweise ist hier ganz analog zur Umwandlung von Hexadezimal nach Integer (s. o.).

```
                  BININT:
04AA 84 63                   STY H1          ;MAXIMALZAHL MERKEN
04AC A0 00                   LDY #0
04AE 84 64                   STY H2
04B0 84 65                   STY H3
                  BININT1:
04B2 C4 63                   CPY H1
04B4 F0 14                   BEQ BININT9
04B6 B9 00 01                LDA PUFFER,Y    ;NAECHSTE ZIFFER
04B9 C9 30                   CMP #$30        ;"0"
04BB 90 0D                   BCC BININT9
04BD C9 31                   CMP #$31        ;CARRY BEACHTEN !
04BF F0 02                   BEQ BININT2
04C1 B0 07                   BCS BININT9
                  BININT2:
04C3 26 65                   ROL H3          ;CARRY REINROTIERN
04C5 26 64                   ROL H2
04C7 C8                      INY
04C8 D0 E8                   BNE BININT1     ;SCHLEIFE
                  BININT9:
04CA 60                      RTS
```

Listing 4/6.3.2-4

Integer nach Binär

Hier ist die Routine analog zur Routine „Integer nach Hexadezimal“ programmiert.

```
                  INTBIN:
04CB A9 00                   LDA #0
04CD 99 00 01                STA PUFFER,Y    ;ENDEMARKIERUNG
04D0 88                      DEY
                  INTBIN1:
04D1 46 64                   LSR H2          ;BIT RAUSSCHIEBEN
04D3 66 65                   ROR H3
04D5 A9 00                   LDA #0
04D7 69 30                   ADC #$30        ;$30 = ASC("0")
04D9 99 00 01                STA PUFFER,Y    ;EINTRAGEN
04DC 88                      DEY
04DD 10 F2                   BPL INTBIN1     ;SCHLEIFE
04DF 60                      RTS
```

Listing 4/6.3.2-5

4/6.3.6

Datenkomprimierung

Autor: Manfred Friese

Wenn Sie auf Ihrem C64 Maschinenprogramme laufen lassen, mehrere Bildschirme nutzen, Berge von Daten berechnen, Speicherbereiche unter den ROMs nutzen oder Graphiken erstellen, stehen Sie sicherlich oft vor dem Problem, einen Speicherbereich auf Diskette speichern oder von Diskette lesen zu wollen. Wenn Sie dies außerdem noch intensiv betreiben, ärgert Sie sicherlich zusätzlich die recht kleine Kapazität der Floppy 1541.

Doch hier bietet sich eine Lösung an: die Datenkomprimierung. Bei der Datenkomprimierung versucht man, bestimmte in dem zu speichernden Bereich vorkommende Bytefolgen durch kürzere zu ersetzen. Eine Methode der Datenkomprimierung wollen wir in diesem Beitrag vorstellen.

Wenn man Datenkomprimierung auf dem 64er betreibt, sollte man aber vorher überlegen, welchen Aufwand man für die Komprimierung treiben will. Der Speicherbereich des 64er ist mit 64 KByte heute als sehr klein zu betrachten. Hinzu kommt die langsame Datenübertragung zur Floppy 1541 über den seriellen Bus.

Daher lohnt sich nur eine einfache Datenkomprimierung. Sie hat den Vorteil, schnell zu sein (so schnell, daß man bei der 1541 nichts davon bemerkt) und sich recht kurz programmieren zu lassen. Was nutzt eine aufwendige Komprimierung, wenn die Routine dazu bereits den halben Speicher belegt?

Wir werden bei unserer Komprimierung versuchen, Folgen gleicher Byte zu finden. Eine Graphik beinhaltet oft große Flächen gleicher Farbe. Diese werden im Speicher durch Byte gleichen Wertes repräsentiert. Zählt man die Bytes gleichen Wertes, kann man eine Folge gleicher Bytes durch ein Byte mit der Anzahl und einem Byte mit dem Wert ersetzen.

Was macht man aber mit Folgen ungleicher Byte? Jedem eine Anzahl voranstellen? Das würde wohl eher zu größeren Dateien führen. Aber es geht auch anders. Wir können ja abzählen, wie viele ungleiche Byte folgen.

Dann müssen wir noch eine Anzahl gleicher von einer Anzahl ungleicher Byte unterscheiden. Für diesen Zweck benutzen wir das oberste Bit des Bytes, in dem die Anzahl gespeichert ist. Ist es gesetzt, folgt eine Sequenz ungleicher Byte, andernfalls eine Sequenz gleicher Byte. In den restlichen sieben Bits können wir dann die Anzahl unterbringen. Ein Byte dieser Form werden wir im folgenden Steuerbyte nennen.

Auf diese Weise lassen sich Sequenzen bis zu 127 Byte kodieren. Längere Sequenzen müssen daher zerlegt werden.

Beispiel

Aus der Bytefolge
1, 2, 3, 4, 5, 5, 5, 5, 5, 5, 6, 7, 8, 9, 10

der Länge 15 Byte wird

132, 1, 2, 3, 4,
6, 5,
133, 7, 8, 9, 10

der Länge 12 Byte.

Mit dieser einfachen Methode sparen wir aber nur etwas, wenn Sequenzen gleicher Byte in dem Bereich vorkommen. Wenn wir Pech haben, gibt es keine solche Sequenz. Dann würde die Datei sogar größer, denn die Datei würde für jeden Block von 127 Byte durch das Hinzukommen des Steuerbytes 128 Byte speichern. Dies entspricht einer Vergrößerung um 0,7 %.

Komplexere Komprimierungsmethoden versuchen daher ganze Bytekombinationen durch kürzere zu ersetzen, was auch bei derartigen Dateien zu Verbesserungen führen kann. Aber auch sie müssen unter Umständen passen. Gute und aufwendige Komprimierungsprogramme versuchen daher verschiedene Methoden und verwenden dann die Methode mit dem kürzesten Ergebnis. Dies kann unter Umständen dann auch die unkomprimierte Fassung sein.

Solche Programme brauchen aber recht lange und werden daher nur zur Archivierung angewandt. Für unsere Zwecke wären sie ungeeignet.

Nachdem wir uns über die Komprimierungsmethode klar geworden sind, können wir uns mit der Gestaltung des Programms befassen. Eine unserer Forderungen war eine angemessene Geschwindigkeit. Dieser Forderung müssen wir natürlich auch bei dem Programm nachkommen. Als Programmiersprache nehmen wir daher den in diesem Buch vorgestellten Assembler.

Bleibt die Frage des Aufrufs der Routinen zu klären. Wir haben uns hier dazu entschlossen, die Routinen als Erweiterung der Befehle *LOAD* und *SAVE* des 64er BASIC zu implementieren. Dazu werden wir eine kleine Routine schreiben, welche die Aufrufe von *LOAD* und *SAVE* erkennt und unsere erweiterten Aufrufe analysiert.

Wenn Sie die ebenfalls in diesem Buch vorgestellte Basic-Erweiterung benutzen, können sie die Routinen natürlich auch dort verwenden.

Die Befehle *LOAD* und *SAVE* sollen bei unserer Implementierung weiter wie bisher arbeiten. Wenn wir die Komprimierung wünschen, signalisieren wir dies mit dem BASIC-Schlüsselwort *ON*. Also lauten die Aufrufe dann

```
LOAD ON "<Dateiname>",<Device>
```

bzw.

```
SAVE ON "<Dateiname>",<Device>
```

Gespeichert wird auch hier wieder ein im Speicher befindliches Programm. Für BASIC-Programme lohnt die Komprimierung aber meist nicht. Nur wenn z.B. Graphiken mit dem *PRINT*-Befehl enthalten sind, kann man akzeptable Komprimierungen erreichen. Außerdem wollen wir ja auch Hires-Graphiken und ähnliches speichern können. Wir brauchen daher noch einen Aufruf, der es uns ermöglicht, beliebige Speicherbereiche auf Diskette zu übertragen. Dieser Aufruf soll die Form

```
SAVE ABS <von>,<bis>,"<Dateiname>",<Device>
```

haben. Die Variable *von* gibt dabei die Startadresse des Speicherbereichs an, und die Variable *bis* die erste nicht mehr zu speichernde Adresse. Wenn Sie z.B. eine Hires-Graphik im Speicherbereich von $A000 bis $BFFF erzeugt haben, wird diese durch den Aufruf

```
SAVE ABS 10*4096,12*4096,"Test",8
```

als Datei *Test* auf Diskette geschrieben. Unser neuer *LOAD*-Befehl lädt Daten immer an ihre Originaladresse. Die gespeicherte Graphik kann daher jederzeit durch den Befehl

```
LOAD ON "Test",8
```

wieder in den Bereich $A000 bis $BFFF geladen werden. Manchmal möchte man jedoch Daten in einen anderen Speicherbereich laden. Auch diese Möglichkeit sollten wir daher vorsehen. Nehmen wir z.B. an, Sie wollen die Graphik der Datei *Test* in den Speicherbereich ab $E000 laden. Genau dies soll dann der Aufruf

```
LOAD ABS 14*4096,"Test",8
```

leisten.

Fassen wir die möglichen Aufrufe noch einmal in einer Tabelle zusammen:

Aufruf	Bedeutung
SAVE "Name",D,S	speichert das aktuelle Programm unter dem Namen *Name* auf das Gerät *D* mit Sekundäradresse *S*. (wie bisher).
SAVE ON "Name",D	speichert das aktuelle Programm komprimiert unter dem Namen *Name* auf das Gerät *D*.
SAVE ABS Von,Bis,"Name",D	speichert den Bereich von *Von* bis *Bis* (ausschließlich) unter dem Namen *Name* auf das Gerät *D*.
LOAD "Name",D,S	wie bisher.
LOAD ON "Name",D	dekomprimiert und lädt die Datei *Name* von Gerät *D* an ihre Originaladresse.
LOAD ABS Von,"Name",D	dekomprimiert und lädt die Datei *Name* von Gerät *D* an die Adresse *Von*.

Bitte beachten Sie dabei, daß bei den Speicher- und Ladebefehlen die ROM-Bereiche immer ausgeblendet werden. Alle Daten beziehen sich daher auf den 64-KByte-RAM-Speicher des C 64.

Wenden wir uns jetzt dem Programmlisting zu. Im folgenden finden Sie das Übersetzungsprotokoll der neuen Lade- und Speicherbefehle. Wir werden sie abschnittweise besprechen.

Das Listing beginnt mit der Vereinbarung der benötigten Konstanten für Zeiger, Betriebssystemroutinen und Vektoren. Als Adresse der Routinen ist $C000 vorgesehen, was Sie selbstverständlich ändern können. Während des Assemblerlaufes erfolgt die Ausgabe der Routinen in das RAM. Sie können anschließend durch den Befehl

```
SYS 12*4096
```

von BASIC aus in das Betriebssystem eingebunden werden.

Die Konstante *Datei* vereinbart die logische Dateinummer für die verwendeten *Open*-Befehle. Sie kann frei gewählt werden. Wir haben hier 13 angegeben.

Die Konstanten *TkLoad* bis *TkAbs* enthalten die Werte der Token für *LOAD* bis *ABS*.

```
** Pass 1 **
Ausgabe auf Diskette? (J=1) 1

** Pass 2 **
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;;; Komprimierungsroutinen  fuer C64 ;;;
        ;;; Version 1.0    (c) M.Friese 6/89 ;;;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

c000                 .ba $c000          ;Startadresse
                     .os                ;Ausgabe in RAM
                     .wa                ;bei Fehler warten
        Routinen:                       ;St artadresse merken

        DATEI:       .eq 13       ;logische Dateinummer f. alle Operationen

        SpNorm:      .eq $37            ;ROM eingeschaltet
        SpRAM:       .eq $34            ;alles RAM
        PP:          .eq 1              ;Prozessorport

        Ymem:        .eq 2              ;Speicher fuer Y-Register
        Integer:     .eq $14            ;Integer-Akku des BASIC
        BasStart:    .eq $2b            ;Startadr. BASIC
        BasEnd:      .eq $2d            ;Endadr. BASIC
        Status:      .eq $90            ;Statusflag IEC-Bus

        azg:         .eq 251            ;allgem. Zeiger 1
        azg2:        .eq 253            ;allgem. Zeiger 2

        GetVec:      .eq $308           ;CHRGET-Vektor

        TkLoad:      .eq 147            ;Token: LOAD
        TkSave:      .eq 148            ;Token: SAVE
        TkOn:        .eq 145            ;Token: ON
        TkAbs:       .eq 182            ;Token: ABS

        ChrGet:      .eq $73            ;CHRGET-Routine
        ChrGot:      .eq $79            ;CHRGOT-Routine

        Open:        .eq $ffc0          ;Datei oeffnen
        Close:       .eq $ffc3          ;Datei schliessen
        ChkOut:      .eq $ffc9          ;Ausgabe in Datei umlenken
        ChkIn:       .eq $ffc6          ;Eingabe aus Datei umlenken
        ClrCh:       .eq $ffcc          ;Ein- und Ausgabe normal
        BasIn:       .eq $ffcf          ;ein Zeichen lesen
        BasOut:      .eq $ffd2          ;ein Zeichen schreiben
        BasLoad:     .eq $e168          ;BASIC-LOAD-Befehl
        BasSave:     .eq $e156          ;BASIC-SAVE-Befehl
        FileParas:   .eq $e1d4          ;Fileparameter lesen
        GetNum:      .eq $ad8a          ;numerischen Wert lesen
        ValInt:      .eq $b7f7          ;Wert nach Integer wandeln
        ChkKom:      .eq $aefd          ;auf Komma testen
```

Listing 4/6.3.6-1

Wenn der erzeugte Code auf Diskette ausgegeben werden soll, wird die Ausgabe in das RAM abgeschaltet. Zusätzlich zu den Routinen erzeugen wir dann noch einen BASIC-Programmkopf, der es uns erlaubt, die Routinen mit einem normalen *LOAD*-Befehl zu laden und mit *RUN* zu starten. Da wir dies bereits bei vielen Gelegenheiten gezeigt haben (z.B. bei dem Assembler selbst), wollen wir hier nicht weiter darauf eingehen.

Dem BASIC-Vorspann folgt noch eine Verschiebeschleife, welche die Routinen nach $C000 verschiebt. Wenn Sie die Startadresse der Routinen ($C000) ändern wollen, brauchen Sie dies nur bei dem *.ba*-Befehl in Zeile 6 zu tun, da diese Adresse durch das Label *Routinen* übernommen wird.

An die Verschiebung schließt sich noch die Ausgabe einer kleinen Meldung an. Durch den Befehl

```
jsr Start
```

werden die Routinen automatisch in das Betriebssystem eingebunden. Sie können dies aber jederzeit noch durch

```
SYS 12*4096
```

erreichen (z.B. nach einem RESET). Durch den *NEW*-Befehl im BASIC-Kopf löscht sich das Programm nach der Initialisierung selbst.

```
               51   ;---------------------------------------
               52   ;-- ggf. Basic-Startprogramm erzeugen --
               53   ;---------------------------------------
               54   DISK:      .in "Ausgabe auf Diskette? (J=1) "
               55              .?e DISK-1          ;Ausgabe auf Disk?
0801           56              .ba $801            ;dann Basis $801
               57              .oc                 ;und keine Ausgabe in's RAM
               58              .ou "@0:kompri"     ;Ausgabe in Datei "kompri"
               59
0801 0d 08 c5  60              .by 13,8,<1989,>1989;1989 SYS2063:NEW
07
0805 9e 32 30  61              .by 158,"2063"      ;erzeugen
36 33
080a 3a a2     62              .by 58,162
080c 00 00 00  63              .by 0,0,0
               64
080f a9 72     65              lda #<ANFANG        ;azg auf Anfang des Codes
0811 a2 08     66              ldx #>ANFANG
0813 85 fb     67              sta azg
0815 86 fc     68              stx azg+1
0817 a9 00     69              lda #<Routinen      ;azg2 auf Zielspeicher
0819 a2 c0     70              ldx #>Routinen
081b 85 fd     71              sta azg2
081d 86 fe     72              stx azg2+1
081f a0 00     73              ldy #0
0821 b1 fb     74   Kopie:     lda (azg),y         ;Speicher kopieren
0823 91 fd     75              sta (azg2),y
0825 e6 fb     76              inc azg             ;azg+1
0827 d0 02     77              bne *+2+2
```

Listing 4/6.3.6-2 (Teil 1)

```
0829 e6 fc    78              inc azg+1
082b e6 fd    79              inc azg2          ;azg2+1
082d d0 02    80              bne *+2+2
082f e6 fe    81              inc azg2+1
0831 a5 fd    82              lda azg2          ;Endwert erreicht ?
0833 c9 0c    83              cmp #<ENDE+1
0835 d0 ea    84              bne Kopie         ;nein
0837 a5 fe    85              lda azg2+1
0839 c9 c2    86              cmp #>ENDE+1
083b d0 e4    87              bne Kopie         ;nein
              88
083d 20 00 c0 89              jsr Start         ;Programm starten
              90
0840 b9 4c 08 91    Melden:   lda Meldung,y     ;Meldung ausgeben
0843 f0 06    92              beq Fertig
0845 20 d2 ff 93              jsr BasOut
0848 c8       94              iny
0849 d0 f5    95              bne Melden
084b 60       96    Fertig:   rts               ;und fertig
              97
              98    ;Text der Meldung
084c 0d cb 4f 99    Meldung:  .by 13,"Komprimierungsroutinen"
4d 50 52 49 4d 49 45 52 55 4e 47 53 52 4f 55 54 49 4e 45 4e
0863 0d 49 4e 100             .by 13,"installiert!",13,0
53 54 41 4c 4c 49 45 52 54 21 0d 00
              101   ANFANG:
              102
              103             .cb Routinen      ;Basisadr. wieder korrigieren
              104             .eb
              105
              106
```

Listing 4/6.3.6-2 (Teil 2)

Die nun folgende *Start*-Routine steht immer an der Adresse $C000 (bzw. an der von Ihnen in Zeile 6 gewählten Adresse). Sie bindet die neuen Befehle in das System ein, indem sie den Vektor der *ChrGet*-Routine auf eine eigene Routine umlenkt. Die *ChrGet*-Routine wird vom Betriebssystem benutzt, um ein Zeichen aus dem BASIC-Programm zu lesen.

```
                   ;-----------------------------------------
                   ;-- Neue Befehle in BASIC einbinden     --
                   ;-----------------------------------------
c000 a9 0b         Start:      lda #<NeuBef        ;CHRGET auf neue Routine
c002 a2 c0                     ldx #>NeuBef        ;umlenken
c004 8d 08 03                  sta GetVec
c007 8e 09 03                  stx GetVec+1
c00a 60                        rts
```

Listing 4/6.3.6-3

Unsere neue Routine ruft zunächst die alte *ChrGet*-Routine auf. Liefert diese das Token für *LOAD* oder für *SAVE*, so arbeiten wir unsere eigenen Befehle ab. Andernfalls überlassen wir alles weitere wie bisher den Routinen des Betriebssystems.

```
c00b 20 73 00      NeuBef:     jsr ChrGet          ;ein Zeichen holen
c00e c9 93                     cmp #TkLoad         ;LOAD-Token ?
c010 f0 0a                     beq MyLoad          ;ja
c012 c9 94                     cmp #TkSave         ;SAVE-Token ?
c014 f0 71                     beq MySave          ;ja
c016 20 79 00                  jsr ChrGot          ;sonst an BASIC zurueckgeben
c019 4c e7 a7                  jmp $a7e7
```

Listing 4/6.3.6-4

Die Routine *MyLoad* wird bei einem *LOAD*-Befehl aufgerufen. Sie testet, ob dem *LOAD*-Token ein Token für *ON* oder für *ABS* folgt, und ruft dann die entsprechenden Routinen auf. Folgt weder ein *ABS* noch ein *ON*, wird einfach die Laderoutine des Betriebssystems gestartet.

```
                 ;-------------------------------------
                 ;-- LOAD-Befehle auswerten          --
                 ;-------------------------------------
c01c 20 73 00    MyLoad:    jsr ChrGet        ;ein Zeiche lesen
c01f c9 b6                  cmp #TkAbs        ;Token ABS ?
c021 f0 0a                  beq LoadAbs       ;ja
c023 c9 91                  cmp #TkOn         ;Token ON ?
c025 f0 36                  beq Load          ;ja
c027 20 68 e1               jsr BasLoad       ;sonst normale Routine aufrufen
c02a 4c ae a7               jmp $a7ae         ;und zu BASIC
```

Listing 4/6.3.6-5

Im folgenden wird der *LOAD-ABS*-Befehl ausgeführt. Wir lesen die Zieladresse als Integer-Zahl (ganzzahlige Zahl zwischen Null und 65535) ein und speichern sie in *azg*. Dann wird über die Betriebssystemroutine *FileParas* Dateiname, Geräteadresse und Sekundäradresse eingelesen. Als logische Dateinummer setzen wir *Datei*. Die evtl. eingelesene Sekundäradresse wird durch Null ersetzt. Dann kann die Datei geöffnet werden.

Durch den Aufruf der *ChkIn*-Routine legen wir fest, daß die Eingabe ab sofort aus der Datei erfolgt. Nachdem wir noch die in der Datei stehende Adresse überlesen haben, können wir bei *LoadEnd* fortfahren.

```
                 ;-------------------------------------
                 ;-- LOAD-ABS-Befehl                 --
                 ;-------------------------------------
c02d 20 73 00    LoadAbs:   jsr ChrGet
c030 20 8a ad               jsr GetNum        ;Adresse einlesen
c033 20 f7 b7               jsr ValInt        ;in Integer wandeln
c036 a5 14                  lda Integer       ;azg auf Adressse
c038 85 fb                  sta azg
c03a a5 15                  lda Integer+1
c03c 85 fc                  sta azg+1
c03e 20 fd ae               jsr ChkKom        ;auf Komma testen
c041 20 d4 e1               jsr FileParas     ;restliche Parameter holen
c044 a9 0d                  lda #DATEI        ;log. Dateinummer
c046 85 b8                  sta $b8           ;setzen
c048 a9 00                  lda #0            ;Sekundaeradr.
```

Listing 4/6.3.6-6 (Teil 1)

```
c04a 85 b9          sta $b9          ;immer Null
c04c 20 c0 ff       jsr Open         ;Open Datei
c04f a2 0d          ldx #DATEI       ;Eingabe aus Datei
c051 20 c6 ff       jsr ChkIn
c054 20 cf ff       jsr BasIn        ;Adresse ueberlesen
c057 20 cf ff       jsr BasIn
c05a 4c 7d c0       jmp LoadEnd      ;weiter bei LoadEnd
```

Listing 4/6.3.6-6 (Teil 2)

Die *LOAD-ON*-Sequenz ist noch einfacher. Hier können wir auf das Einlesen der Integer-Zahl verzichten. Die Adresse, die wir bei *LOAD ABS* überlesen haben, speichern wir statt dessen in *azg*. Dann rufen wir die Ladeschleife *LoadLoop* auf. Sie lädt die Daten in den Speicher. Beendet wird der Befehl wieder von dem Betriebssystem. Es erwartet für seine Arbeit in X und Y die Endadresse des Speicherbereichs.

```
                ;-------------------------------------
                ;-- LOAD-ON-Befehl                  --
                ;-------------------------------------
c05d 20 73 00   Load:     jsr ChrGet
c060 20 d4 e1             jsr FileParas    ;Parameter holen
c063 a9 0d                lda #DATEI       ;log. Dateinummer
c065 85 b8                sta $b8          ;setzen
c067 a9 00                lda #0           ;Sekundaeradr.
c069 85 b9                sta $b9          ;immer Null
c06b 20 c0 ff             jsr Open         ;Open Datei
c06e a2 0d                ldx #DATEI       ;Eingabe aus Datei
c070 20 c6 ff             jsr ChkIn
c073 20 cf ff             jsr BasIn        ;Adresse in azg
c076 85 fb                sta azg          ;einlesen
c078 20 cf ff             jsr BasIn
c07b 85 fc                sta azg+1
c07d 20 97 c1   LoadEnd:  jsr LoadLoop     ;Ladeschleife
c080 a6 fb                ldx azg          ;Endadr. in X/Y
c082 a4 fc                ldy azg+1
c084 4c a1 e1             jmp $e1a1        ;weiter beim BASIC-LOAD
```

Listing 4/6.3.6-7

Bitte beachten Sie, daß der *LOAD-ON*- bzw. *LOAD-ABS*-Befehl genau wie der normale *LOAD*-Befehl abgeschlossen wird. Dies bedeutet insbesondere, daß ein Basic-Programm von vorn abgearbeitet wird. Ist dies nicht erwünscht, sollten Sie die Zeile 180 durch den Befehl

```
jmp $a7ae
```

einsetzen. Die Zeilen 178 und 179 können dann entfallen. Allerdings wird dann auch nicht mehr das Speicherende beim Laden von komprimierten Basic-Programmen richtig gesetzt.

Die Routine *MySave* arbeitet wie *MyLoad*. Sie wird bei einem *SAVE*-Befehl aufgerufen. Sie testet, ob dem *SAVE*-Token ein Token für *ON* oder für *ABS* folgt, und ruft dann die entsprechenden Routinen auf. Folgt weder ein *ABS* noch ein *ON*, wird wieder einfach die Speicherroutine des Betriebssystems gestartet.

```
              183  ;--------------------------------------
              184  ;-- SAVE-Befehle auswerten            --
              185  ;--------------------------------------
c087 20 73 00 186  MySave:    jsr ChrGet          ;ein Zeichen lesen
c08a c9 b6    187             cmp #TkAbs          ;ABS-Token ?
c08c f0 0a    188             beq SaveAbs         ;ja
c08e c9 91    189             cmp #TkOn           ;ON-Token
c090 f0 32    190             beq Save            ;ja
c092 20 56 e1 191             jsr BasSave         ;sonst normale SAVE-Routine
c095 4c ae a7 192             jmp $a7ae           ;zurueck zu Basic
              193
              194
```

Listing 4/6.3.6-8

Bei dem *SAVE-ABS*-Befehl lesen wir zunächst die Startadresse ein und speichern sie in *Von*. Dann muß ein Komma folgen. Die nun folgende Endadresse wird wieder eingelesen und in *Bis* abgelegt. Dann können wir unsere Arbeit bei *DoSave* fortsetzen.

```
                 ;------------------------------------
                 ;-- SAVE-ABS-Befehl                 --
                 ;------------------------------------
c098 20 73 00    SaveAbs:  jsr ChrGet
c09b 20 8a ad              jsr GetNum     ;Adresse lesen
c09e 20 f7 b7              jsr ValInt     ;in Integer wandeln
c0a1 a5 14                 lda Integer    ;als VON speichern
c0a3 8d 07 c2              sta von
c0a6 a5 15                 lda Integer+1
c0a8 8d 08 c2              sta von+1
c0ab 20 fd ae              jsr ChkKom     ;auf Komma testen
c0ae 20 8a ad              jsr GetNum     ;Adresse lesen
c0b1 20 f7 b7              jsr ValInt     ;in Integer wandeln
c0b4 a5 14                 lda Integer
c0b6 8d 09 c2              sta bis        ;als BIS speichern
c0b9 a5 15                 lda Integer+1
c0bb 8d 0a c2              sta bis+1
c0be 20 fd ae              jsr ChkKom     ;auf Komma testen
c0c1 4c db c0              jmp DoSave     ;weiter bei DoSave
```

Listing 4/6.3.6-9

Noch einfacher ist es bei dem *SAVE-ON*-Befehl. Hier brauchen wir keine Adressen einzulesen, sondern wir kopieren einfach die BASIC-Startadresse in *Von* und die BASIC-Endadresse in *Bis*.

Dann werden die restlichen Dateiparameter gelesen. Wie bei den *LOAD*-Befehlen wird die logische Dateinummer gesetzt und eine evtl. vorhandene Sekundäradresse ersetzt. Dann kann die Datei geöffnet werden. Durch den folgenden *ChkOut*-Aufruf wird jede nun folgende Ausgabe in die Datei umgeleitet.

Die Routine *SaveLoop* speichert den angegebenen Bereich dann in die Datei. Anschließend können wir die Kontrolle wieder an das Betriebssystem übergeben.

```
                 ;------------------------------------
                 ;-- SAVE-ON-Befehl                  --
                 ;------------------------------------
c0c4 a5 2b       Save:     lda BasStart   ;VON ist Basic-
c0c6 8d 07 c2              sta von        ;Startadresse
c0c9 a5 2c                 lda BasStart+1
c0cb 8d 08 c2              sta von+1
c0ce a5 2d                 lda BasEnd     ;BIS ist Basic-
c0d0 8d 09 c2              sta bis        ;Endadresse
```

Listing 4/6.3.6-10 (Teil 1)

```
c0d3 a5 2e               lda BasEnd+1
c0d5 8d 0a c2            sta bis+1
c0d8 20 73 00            jsr ChrGet
c0db 20 d4 e1  DoSave:   jsr FileParas     ;Fileparameter lesen
c0de a9 0d               lda #DATEI        ;log. Dateinummer
c0e0 85 b8               sta $b8           ;setzen
c0e2 a9 01               lda #1            ;Sekundaeradr.
c0e4 85 b9               sta $b9           ;ist immer 1
c0e6 20 c0 ff            jsr Open          ;Open Datei
c0e9 a2 0d               ldx #DATEI        ;Ausgabe in Datei
c0eb 20 c9 ff            jsr ChkOut
c0ee ad 07 c2            lda von           ;Startadr. in Datei schreiben
c0f1 85 fb               sta azg
c0f3 20 d2 ff            jsr BasOut
c0f6 ad 08 c2            lda von+1
c0f9 85 fc               sta azg+1         ;und in azg bringen
c0fb 20 d2 ff            jsr BasOut
c0fe 20 04 c1            jsr SaveLoop      ;Speicherschleife
c101 4c ae a7            jmp $a7ae         ;und weiter im BASIC
```

Listing 4/6.3.6-10 (Teil 2)

Bis jetzt haben wir nur vorbereitende Arbeiten erledigt. Mit Datenkomprimierung oder Datendekomprimierung hatte das nichts zu tun.

Die Datenkomprimierung erfolgt in der hier folgenden Speicherschleife *SaveLoop*. Um sie zu verstehen, sollten Sie die Funktion der Routinen *GetAZG, CmpAZG* und *PutAZG* kennen. Sie werden weiter unten beschrieben.

SaveLoop versucht zunächst eine möglichst lange Sequenz gleicher Byte zu finden. Spätestens nach 127 Byte muß jedoch abgebrochen werden, da für längere Sequenzen kein Steuerbyte mehr erzeugt werden kann. Die Länge der Sequenz befindet sich dann im Y-Register.

Falls die Endadresse *Bis* durch die Sequenz überschritten wird, reduziert der Aufruf von *SaEnde?* die Länge auf das zulässige Maß.

Bei mehr als zwei gleichen Byte wird ein Steuerbyte erzeugt und zusammen mit dem Byte in die Datei geschrieben. Dann wird auf *azg* die Länge der Sequenz addiert. Haben wir die Adresse *Bis* erreicht, können wir die Datei schließen und zur aufrufenden Routine zurückkehren. Andernfalls beginnt *SaveLoop* von vorn.

Finden wir weniger als zwei gleiche Byte, suchen wir die maximale Sequenz ungleicher Byte. Natürlich darf auch diese höchstens 127 Byte lang sein und die Endadresse nicht überschreiten.

Dann schreiben wir ein Steuerbyte in die Datei, gefolgt von der Sequenz der ungleichen Byte. Anschließend wird wieder die Länge der Sequenz auf *azg* addiert und mit der Adresse *Bis* verglichen. Haben wir die Endadresse erreicht, können wir die Datei schließen und zur aufrufenden Routine zurückkehren. Andernfalls beginnt *SaveLoop* wieder von vorn.

```
                ;-----------------------------------
                ;-- SAVE-Speicherschleife          --
                ;-----------------------------------
c104 a0 00      SaveLoop:  ldy #0           ;ab azg folgende Byte
c106 20 d4 c1              jsr GetAZG       ;untersuchen
c109 c8         SuchGleich:iny
c10a 20 e3 c1              jsr CmpAZG       ;gleiche Bytes ?
c10d d0 04                 bne EndGleich    ;nein
c10f c0 7e                 cpy #126         ;schon 126 gleiche Byte?
c111 d0 f6                 bne SuchGleich   ;nein, weitervergleichen

c113 20 7f c1   EndGleich: jsr SaEnde?      ;BIS ueberschritten ?
c116 c0 02                 cpy #2           ;mehr als 2 gleiche ?
c118 b0 38                 bcs Gleiche      ;ja
c11a a0 ff                 ldy #$ff         ;sonst Sequenz Ungleiche testen
c11c 84 02                 sty Ymem
c11e e6 02      SuchUngl:  inc Ymem
c120 a4 02                 ldy Ymem
c122 c0 7f                 cpy #127         ;schon 127 Ungleiche ?
c124 f0 0f                 beq ZuLang       ;ja
c126 20 d4 c1              jsr GetAZG       ;beginnt eine Sequenz
c129 c8                    iny              ;Gleiche ?
c12a 20 e3 c1              jsr CmpAZG
c12d d0 ef                 bne SuchUngl     ;nein
c12f c8                    iny
c130 20 e3 c1              jsr CmpAZG
c133 d0 e9                 bne SuchUngl     ;nein

c135 a4 02      ZuLang:    ldy Ymem         ;Sequenz Ungleich abschliessen
c137 20 7f c1              jsr SaEnde?      ;BIS ueberschritten ?
c13a 84 02                 sty Ymem         ;Laenge der Sequenz merken
c13c 98                    tya
c13d 48                    pha
c13e 09 80                 ora #128         ;Bit 7 setzen
c140 20 d2 ff              jsr BasOut       ;und ausgeben
c143 a0 00                 ldy #0
c145 20 d4 c1   OutLoop:   jsr GetAZG       ;Sequenz der Ungl.
c148 20 d2 ff              jsr BasOut       ;ausgeben
```

Listing 4/6.3.6-11 (Teil 1)

```
c14b c8                         iny
c14c c4 02                      cpy Ymem        ;bis alle Byte ausgegeben
c14e d0 f5                      bne OutLoop
c150 f0 0d                      beq add         ;azg neu setzen

c152 98          Gleiche:       tya             ;Sequenz Gleiche schreiben
c153 48                         pha             ;Laenge merken
c154 20 d2 ff                   jsr BasOut      ;und ausgeben
c157 a0 00                      ldy #0          ;Zeichen holen
c159 20 d4 c1                   jsr GetAZG
c15c 20 d2 ff                   jsr BasOut      ;und ausgeben
c15f 68          add:           pla             ;Laenge zurueckholen
c160 18                         clc             ;und auf azg addieren
c161 65 fb                      adc azg
c163 85 fb                      sta azg
c165 90 02                      bcc SaWeiter
c167 e6 fc                      inc azg+1

c169 a5 fb       SaWeiter:      lda azg         ;BIS erreicht ?
c16b cd 09 c2                   cmp bis
c16e d0 94                      bne SaveLoop    ;nein
c170 a5 fc                      lda azg+1
c172 cd 0a c2                   cmp bis+1
c175 d0 8d                      bne SaveLoop    ;nein

c177 20 cc ff                   jsr ClrCh       ;Ausgabe normal
c17a a9 0d                      lda #DATEI      ;Datei schliessen
c17c 4c c3 ff                   jmp Close
```

Listing 4/6.3.6-11 (Teil 2)

Die Routine *SaEnde?* testet, ob die Endadresse durch eine Sequenz überschritten wird. Dazu erhält sie die Adresse des Anfangs der Sequenz in *azg* und die Länge der Sequenz im Y-Register.

Durch Subtraktion der Anfangsadresse der Sequenz von der Endadresse des Speicherbereiches in *Bis* erhalten wir die maximale Länge einer noch im Bereich liegenden Sequenz. Ist das Highbyte verschieden von Null, so paßt eine Sequenz mit Länge in Y immer. Wir ändern daher nichts.

Sonst vergleichen wir Y mit dem Lowbyte. Ist Y kleiner oder gleich, brauchen wir ebenfalls nicht zu korrigieren. Sonst ersetzen wir den Wert in Y durch das Lowbyte, was genau der Länge bis zur Endadresse entspricht.

```
                  ;--------------------------------------
                  ;-- BIS Ueberschritten ?????          --
                  ;--------------------------------------
c17f 38           SaEnde?:   sec               ;BIS-AZG berechnen
c180 ad 09 c2                lda bis
c183 e5 fb                   sbc azg
c185 85 fd                   sta azg2          ;Low-Byte in azg2
c187 ad 0a c2                lda bis+1
c18a e5 fc                   sbc azg+1
c18c d0 08                   bne ok            ;High Byte<>0, dann alles ok
c18e c4 fd                   cpy azg2          ;Y groesser als Low-Byte ?
c190 90 04                   bcc ok
c192 f0 02                   beq ok
c194 a4 fd                   ldy azg2          ;ja, dann durch Low-Byte ersetzen
c196 60           ok:        rts
```

Listing 4/6.3.6-12

Bei der Ladeschleife *LoadLoop* haben wir es leicht. Wir lesen immer ein Byte aus der Datei ein. Ist es positiv (Bit 7 nicht gesetzt), so gibt es an, wie oft wir das nachfolgende Byte speichern müssen. Wir brauchen also nur das nachfolgende Byte zu lesen und so oft zu speichern, wie es das Steuerbyte angab.

War das Steuerbyte hingegen negativ, setzen wir Bit 7 auf Null, so daß wir die Anzahl der zu lesenden ungleichen Byte erhalten. Diese werden dann aus der Datei gelesen und gespeichert.

Anschließend testen wir die Statusvariable *Status*. Ist sie Null, stehen noch Byte zur Verfügung. *LoadLoop* beginnt dann von vorn. Andernfalls haben wir das Dateiende erreicht und können unsere Arbeit beenden.

```
                  ;--------------------------------------
                  ;-- LOAD-Ladeschleife                --
                  ;--------------------------------------
c197 20 cf ff     LoadLoop:  jsr BasIn         ;ein Zeichen aus Datei lesen
c19a c9 00                   cmp #0            ;Bit7gesetzt?
c19c 30 0e                   bmi LoUngl        ;ja, Ungleiche laden
c19e a8                      tay               ;sonst Laenge merken
c19f aa                      tax
c1a0 20 cf ff                jsr BasIn         ;Zeichen lesen
c1a3 88           LoGleich:  dey               ;und Y-mal speichern
c1a4 30 18                   bmi LoWeiter
c1a6 20 f6 c1                jsr PutAZG
c1a9 4c a3 c1                jmp LoGleich
```

Listing 4/6.3.6-13 (Teil 1)

```
c1ac 29 7f     LoUngl:    and #%1111111    ;Bit 7 auf 0 setzen
c1ae 85 02                sta Ymem         ;ergibt Laenge
c1b0 aa                   tax              ;Laenge merken
c1b1 a0 00                ldy #0
c1b3 20 cf ff  UnglLoop:  jsr BasIn        ;Y-Mal ein Zeichen lesen
c1b6 20 f6 c1             jsr PutAZG       ;und speichern
c1b9 c8                   iny
c1ba c4 02                cpy Ymem
c1bc d0 f5                bne UnglLoop

c1be 8a        LoWeiter:  txa              ;Laenge auf azg
c1bf 18                   clc              ;addieren
c1c0 65 fb                adc azg
c1c2 85 fb                sta azg
c1c4 90 02                bcc NoHi2
c1c6 e6 fc                inc azg+1
c1c8 a5 90     NoHi2:     lda Status       ;Status=0
c1ca f0 cb                beq LoadLoop     ;ja, weiter

c1cc 20 cc ff             jsr ClrCh        ;sonst Eingabe normal
c1cf a9 0d                lda #DATEI       ;und Datei schliessen
c1d1 4c c3 ff             jmp Close
```

Listing 4/6.3.6-13 (Teil 2)

Jetzt fehlen nur noch die Routinen *GetAZG, CmpAZG* und *PutAZG*. Sie arbeiten analog zu einfachen Assemblerbefehlen. Ihre Funktion entnehmen Sie daher bitte der folgenden Tabelle:

Routine	arbeitet wie
GetAZG	lda (azg),y
PutAZG	sta (azg),y
CmpAZG	cmp (azg),y

Im Unterschied zu den Assemblerbefehlen wird jedoch vor der Aktion der gesamte Speicher auf RAM umgeschaltet. Dann wird die Aktion durchgeführt und anschließend wieder die normale Speicheraufteilung eingeschaltet.

```
                ;------------------------------------
                ;-- Speichern, Lesen von Vergleichen  --
                ;-- unter, von bzw. mit (azg),y       --
                ;------------------------------------
c1d4 78         GetAZG:    sei                ;kein Interrupt
c1d5 a9 34                 lda #SpRAM         ;alles RAM
c1d7 85 01                 sta PP
c1d9 b1 fb                 lda (azg),y        ;Wert lesen
c1db 48                    pha                ;und merken
c1dc a9 37                 lda #SpNorm        ;Speicher normal
c1de 85 01                 sta PP
c1e0 68                    pla                ;Wert in Akku
c1e1 58                    cli                ;Interrupt zulassen
c1e2 60                    rts

c1e3 78         CmpAZG:    sei                ;kein Interrupt
c1e4 48                    pha                ;Wert merken
c1e5 a9 34                 lda #SpRAM         ;alles RAM
c1e7 85 01                 sta PP
c1e9 68                    pla
c1ea d1 fb                 cmp (azg),y        ;Wert vergleichen
c1ec 08                    php                ;Ergebnis merken
c1ed 48                    pha                ;Wert merken
c1ee a9 37                 lda #SpNorm        ;Speicher normal
c1f0 85 01                 sta PP
c1f2 68                    pla                ;Wert in Akku
c1f3 28                    plp                ;Ergebnis
c1f4 58                    cli                ;Interrupt zulassen
c1f5 60                    rts

c1f6 78         PutAZG:    sei                ;kein Interrupt
c1f7 48                    pha                ;Wert merken
c1f8 48                    pha
c1f9 a9 34                 lda #SpRAM         ;alles RAM
c1fb 85 01                 sta PP
c1fd 68                    pla
c1fe 91 fb                 sta (azg),y        ;Wert speichern
c200 a9 37                 lda #SpNorm        ;Speicher normal
c202 85 01                 sta PP
c204 68                    pla                ;Akku wieder auf Wert
c205 58                    cli                ;Interrupt zulassen
c206 60                    rts


                ;------------------------------------
                ;-- Speicher fuer VON und BIS        --
                ;------------------------------------
                von:       .dw 1
                bis:       .dw 1
                ENDE:
```

Listing 4/6.3.6-14

An dieser Stelle drucken wir noch das für eine Fehlersuche sehr hilfreiche Label-File ab:

Label-File

add	p c15f	azg	f 00fb	azg2	f 00fd	bis	p c209
ok	p c196	von	p c207	ANFANG	p 0872	BasEnd	f 002d
BasIn	f ffcf	BasLoad	f e168	BasOut	f ffd2	BasSave	f e156
BasStart	f 002b	ChkIn	f ffc6	ChkKom	f aefd	ChkOut	f ffc9
ChrGet	f 0073	ChrGot	f 0079	Close	f ffc3	ClrCh	f ffcc
CmpAZG	p c1e3	DoSave	p c0db	DATEI	f 000d	DISK	f 0001
EndGleich	p c113	ENDE	p c20b	Fertig	p 084b	FileParas	f e1d4
GetAZG	p c1d4	GetNum	f ad8a	GetVec	f 0308	Gleiche	p c152
Integer	f 0014	Kopie	p 0821	Load	p c05d	LoadAbs	p c02d
LoadEnd	p c07d	LoadLoop	p c197	LoGleich	p c1a3	LoUngl	p c1ac
LoWeiter	p c1be	Melden	p 0840	Meldung	p 084c	MyLoad	p c01c
MySave	p c087	NeuBef	p c00b	NoHi2	p c1c8	Open	f ffc0
OutLoop	p c145	PutAZG	p c1f6	PP	f 0001	Routinen	p c000
Save	p c0c4	SaveAbs	p c098	SaveLoop	p c104	SaEnde?	p c17f
SaWeiter	p c169	SpNorm	f 0037	SpRAM	f 0034	Start	p c000
Status	f 0090	SuchGleich	p c109	SuchUngl	p c11e	TkAbs	f 00b6
TkLoad	f 0093	TkOn	f 0091	TkSave	f 0094	UnglLoop	p c1b3
ValInt	f b7f7	Ymem	f 0002	ZuLang	p c135		

4/6.3.9

Text-Convert-System

4/6.3.9.1

Bedienung von TCS
Autor: Matthias Groß

Einleitung

Was kann das TCS?

Will man in einem Basic-Programm Texte auf den Bildschirm bringen, so ist das ganz einfach: man benutzt den PRINT-Befehl. Für komplizierte Bildschirmmasken gibt es zahlreiche Editoren, die eingegebene Texte in Basic-Zeilen umwandeln. Doch man stößt schnell an die Grenzen dieser Methode. Der PRINT-Befehl ist zu langsam, beim Hintereinanderschalten von mehreren Bildschirmseiten entsteht ein starkes Flimmern.

Ein Wechsel zur Maschinensprache bringt hier auch keine Besserung. Es gibt zwar Routinen zur Textausgabe von ASCII-Zeichen, doch diese arbeiten nicht wesentlich schneller als von Basic aus. Besser wäre es also, die Zeichencodes direkt in den Bildschirm zu schreiben. Davon abgesehen, daß diese Möglichkeit nur von wenigen Assemblern unterstützt wird, existieren hier keine Editoren zum Eingeben der Maske.

Aus diesem Problem heraus entstand das Text-Convert-System. Mit dem TCS-Converter kann der aktuelle Bildschirminhalt eingefroren und gepackt im Speicher abgelegt werden. Mit einer Basicerweiterung können bis zu 255 solcher Texte dann blitzschnell wieder zurück auf den Bildschirm geholt werden.

Die Arbeitsweise des TCS

Der Umgang mit dem TCS läßt sich am besten an einem Beispiel verdeutlichen. Nehmen wir an, Sie wollen aus einem Basic-Programm heraus einige Bildschirme ins TCS-Format konvertieren. Dazu gehen Sie folgendermaßen vor:

1. Laden Sie den Converter mit <LOAD"TCS CONVERTER",8,1> und starten Sie ihn mit <SYS 39300>.

2. Sie laden nun das Programm, das die Masken enthält, und bringen eine davon auf den Bildschirm. Dann drücken Sie zuerst <CTRL> und <Pfeil links>, dann die <RETURN>-Taste, und der Bildschirm ist gepackt im Speicher abgelegt. So auch mit den anderen Bildschirmen verfahren.

3. Die Bildschirme mit <DISK"<Name>,P,W",40960> auf Diskette speichern.

4. Jetzt können die Masken eingesetzt werden. Dazu laden Sie den Deconverter (LOAD"TCS DECONVERTER",8 & RUN). Dann laden Sie die vorher abgespeicherten Texte mit <DISK"NAME">. Jetzt können Sie mit dem Befehl <BS <Nr. 1-255> > die Masken zurück auf den Bildschirm holen und so in Ihrem Programm einsetzen.

Um die Befehlserweiterung mit Ihrem Programm abzuspeichern (ohne sie sind die Befehle nicht lauffähig), genügt die Eingabe von <NRM> und dann entsprechend <SAVE. . .>.

In den folgenden Kapiteln finden Sie eine genaue Beschreibung der Befehle und Funktionen des TCS.

Befehlsbeschreibung Converter

Allgemein

Der Converter wird mit <LOAD"TCS CONVERTER",8,1> eingeladen und mit <SYS39300> gestartet. Im Folgenden finden Sie eine Beschreibung der neuen Befehle, die Ihnen dieser Programmteil zur Verfügung stellt.

Hinter den eigentlichen Befehlsnamen stehen Begriffe, von denen der Befehl abgeleitet wurde. Anschließend wird die genaue Syntax angegeben. Alle Parameter können auch durch Variablen oder entsprechende Ausdrücke angegeben werden. Die Befehle lassen sich nicht abkürzen!

Wird ein TCS-Befehl direkt hinter einer IF . . . THEN-Bedingung verwendet, muß ein Doppelpunkt vorangestellt werden.

Beispiel:

```
100 IF A=1 THEN:BS 5
```

1. INIT (Initialisierung)

```
SYNTAX: INIT (keine Parameter)
```

Dieser Befehl wird beim ersten Start des Converters angesprungen, um eine Sprungtabelle im Speicher anzulegen. Jeder weitere Aufruf dieses Befehls löscht die Sprungtabelle und somit die im Speicher befindlichen Bildschirme. Anschließend wird der Befehl 'TAST 1' ausgeführt.

2. START (Anfang)

```
SYNTAX: START <Speicheradresse>
```

Dieser Befehl legt fest, ab welcher Speicherstelle die Texte vorläufig abgelegt werden sollen. Es kann dabei auch das RAM unter dem ROM benützt werden. Der Einschaltwert ist $A000 (40960). Man sollte vorsichtig mit diesem Befehl umgehen, damit man keine Adressen in der Zeropage oder dem Basic-Programm angibt, denn anschließend wird der Befehl INIT ausgeführt.

3. TAST (Tastaturvektor an/aus)

```
SYNTAX: TAST <Argument 0-255>
```

Dieser Befehl ist für den Tastaturvektor verantwortlich. Ist das Argument 0, wird er ausgeschaltet. Die Tastenkombination RUNSTOP/RESTORE schaltet ihn ebenfalls ab. Bei jedem anderen Argument (1-255) wird der Vektor auf die TCS-Routine umgestellt.

Ist der Tastaturvektor eingeschaltet, wird mit der Tastenkombination <CTRL> + <Pfeil links> der Converter aufgerufen. Im Bildschirmrahmen sind jetzt hell- und dunkelblaue Streifen zu sehen: Der Rechner befindet sich im Wartezustand. Zwei Tasten haben jetzt noch eine Wirkung:

SPACE (Leertaste): Das ist die Exitfunktion. Der C 64 springt zurück in den Direkt-/Programmodus.

RETURN: Die Convertierroutine. Alles, was in diesem Moment auf dem Textbildschirm zu sehen ist, wird in den Computerspeicher übertragen. Danach wird wieder in den Direkt-/Programmodus gesprungen. Der Packvorgang dauert maximal eine halbe Sekunde.

4. SCREEN (Bildschirmparameter)

SYNTAX: SCREEN <Bildschirmanfang>

Normalerweise liegt der Bildschirmspeicher ab 1024. Für bestimmte Anwendungen ist es günstiger, den Bildschirm zu verschieben. Mit diesem Befehl teilen Sie dem TCS die neue Adresse mit.

5. COL (Farbe)

SYNTAX: COL <Argument 0-255>

Das TCS ist in der Lage, zusammen mit dem Bildschirminhalt auch noch die Hintergrund- und Rahmenfarbe abzuspeichern. Ist das Argument größer oder gleich 1, wird diese Funktion eingeschaltet. <COL 0> schaltet die Option ab. Einschaltwert ist 0.

6. MEM (Memory)

SYNTAX: MEM (keine Parameter)

Dieser Befehl gibt zwei Zahlen aus: den Anfang der Texte im Speicher (siehe auch <START>) und die laufende Adresse. Ab dieser Adresse wird der nächste Bildschirm abgelegt. Wollen Sie aus diesen Angaben die Länge der bis jetzt gespeicherten Texte berechnen, so beachten Sie bitte, daß vor den eigentlichen Bildschirmen eine 512 Byte lange Sprungtabelle steht, die später nur noch teilweise benötigt wird.

7. NUM (Nummer, Anzahl)

SYNTAX: NUM (keine Parameter)

NUM gibt die Anzahl der bisher im Speicher befindlichen Bildschirme aus.

8. DISK (Diskettenoperation)

SYNTAX: DISK "<Name>,P,W", <spätere Startadresse>

Dieser Befehl sichert die momentan im Speicher befindlichen Texte in einem speziellen Format auf Diskette, die vom Deconverter dann wieder eingelesen werden können. Der Parameter <spätere Startadresse> gibt an, ab welcher Adresse die Texte später im Speicher liegen sollen.

9. OFF (Ausschalten)

SYNTAX: OFF (keine Parameter)

Mit diesem Befehl wird die gesamte Erweiterung abgeschaltet. Ein Neustart ist mit <SYS 39300> möglich. Die alten Parametereinstellungen bleiben dabei erhalten.

Befehlsbeschreibung Deconverter

Der Deconverter wird mit <LOAD"TCS DECONVERTER",8> geladen. Nach dem Start wird der Basicanfang hinter das Programm verschoben und der Befehl <RUN> ausgeführt.

1. NRM (normal)

SYNTAX: NRM (keine Parameter)

NRM stellt die Basiczeiger auf den Anfang des Deconverters. Jetzt kann mittels SAVE-Befehl das eigentliche Programm zusammen mit dem Deconverter-Vorspann abgespeichert werden.

2. PRG (Programm)

SYNTAX: PRG (keine Parameter)

Dieser Befehl liegt den Basicanfang zurück auf das Programm.

3. DISK (Diskettenoperationen)

SYNTAX: DISK "<Name>"

Analog zum DISK-Befehl des Converters, der die Texte abspeichert, lädt der Befehl hier die TCS-Files wieder in den Computerspeicher zurück. Handelt es sich bei dem angegebenen Namen nicht um ein TCS-File, wird ein <NO TCS SCREEN ERROR> ausgegeben und der Ladevorgang abgebrochen. Während des Programmablaufs darf man beliebig viele Textfiles nachladen.

4. BS (Bildschirm)

```
SYNTAX: BS <BS-Nr. X (1-255)
```

Dieser Befehl sorgt dafür, daß der Bildschirm Nr. X ausgegeben wird. Ist X=0 oder größer als die Anzahl der eingeladenen Bildschirme, wird die Routine mit einem <ILLEGAL QUANTITY ERROR> abgebrochen.

Trifft das Programm beim Ausdrucken des Screens auf ein Zeichen, das nicht vom TCS-Converter stammen kann, wird das mit einem <SCREEN ERROR> quittiert. Sie haben dann wahrscheinlich den von Texten belegten Speicherbereich durch POKEs oder ein Maschinensprache-Programm überschrieben.

5. SCREEN

Siehe SCREEN-Befehl des Converters.

6. NUM

Siehe NUM-Befehl des Converters.

7. MEM

Hier wird aber die exakte Anfangs- und Endadresse der Texte im Speicher angezeigt.

8. SPACE

```
SYNTAX: SPACE <X (0-255)>
```

<SPACE> legt die Farbe der Leerzeichen fest, voreingestellt ist Null.

9. BASIC

SYNTAX: BASIC (<X (0-65535)>)

Mit diesem Befehl läßt sich der Anfang des Basicspeichers verschieben. Von der angegebenen Adresse wird eins abgezogen und die sich daraus ergebende Speicherstelle auf null gesetzt. Nach der Ausführung des Befehls empfiehlt sich die Eingabe von <NEW>. <BASIC> ohne Parameter verlegt den Basicspeicher ans Ende der zuletzt eingeladenen Textfiles.

10. OFF

Ein Wiedereinschalten erfolgt mit <SYS 2084>.

Besonderheiten des TCS

Dieses Kapitel ist als Ergänzung zu den Befehlsbeschreibungen gedacht und soll den Umgang mit den neuen Funktionen sowie deren Eigenheiten erläutern.

1. Unterschiedliche Leerzeichen

Der Bildschirmcode für ein Leerzeichen ist normalerweise 32. Es gibt aber auch noch einen weiteren Code, dem ein Space zugeteilt ist: Nr. 96. Beim Editieren am Bildschirm können diese Codes gemischt vorkommen. Der Benutzer merkt davon nichts, die Zeichen sind ja identisch. Für den Converter besteht darin jedoch ein Problem, denn er kann z.B. gleiche Zeichenfolgen nur dann zusammenfassen, wenn deren Codes auch wirklich gleich sind. Aus diesem Grund wurden die beiden Zeichen gleichgesetzt: überall, wo der Converter auf den Code 96 trifft, ersetzt er ihn durch den Code 32.

Diese Eigenheit des TCS ist im Normalfall unerheblich. Wenn Sie aber einen neuen Zeichensatz verwenden, sollten Sie das Zeichen Nr. 96 nicht benützen, denn es taucht in einem gepackten Bildschirm nicht mehr auf.

2. Die Farbe der Leerzeichen

Treten beim PRINT-Befehl Leerzeichen auf, so erhalten diese die aktuelle Zeichenfarbe. Dies ist nur dann wichtig, wenn Sie in die Bildschirmmaske nachträglich noch Zeichen „POKEn“ wollen. Das TCS berücksichtigt die Farbe der Leerzeichen

nicht, man kann sie deshalb nachträglich ändern: Der bei <SPACE> angegebene Parameter wird als Farbcode interpretiert und beim nächsten BS-Befehl verwendet.

Wird als Parameter eine Zahl größer 16 angegeben, so wird der Bildschirminhalt nicht gelöscht, sondern die Maske über den aktuellen Bildschirm „gelegt".

Mit <SPACE 16> wird eine Sonderfunktion des TCS eingeschaltet: Der „Window-Modus". Dabei werden beim BS-Befehl nur die Leerzeichen des Bildschirms gelöscht, die zwischen dem ersten und letzten Textzeichen stehen. Dieser Modus ist speziell für die Unterstützung von Textfenstern gedacht. Der Hintergrund des Fensters wird gelöscht, die restlichen Zeichen bleiben davon unberührt (siehe Bild).

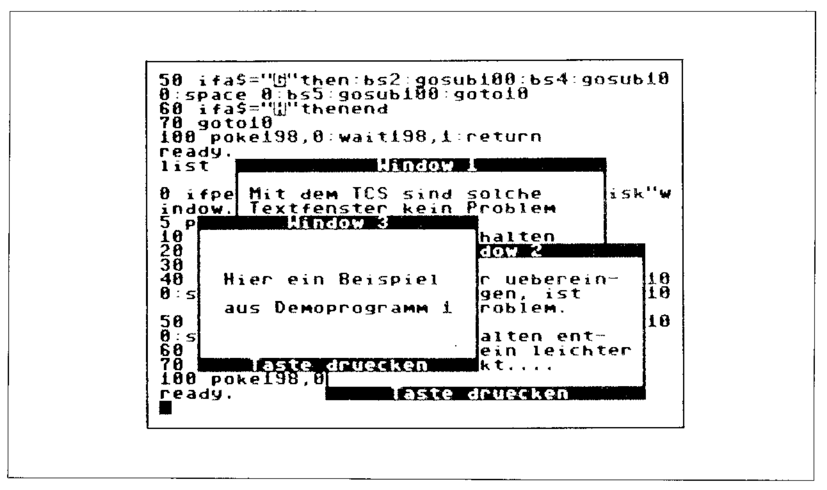


Bild 4/6.3.9.1

3. Speicherorganisation

Zum Speichern der Bildschirme steht dem Benutzer unter TCS prinzipiell der gesamte RAM-Bereich des C 64 zur Verfügung. Deshalb wird man seine Texte gewöhnlich unter das Basic-ROM ab $A000 legen. Der Nachteil hierbei: Die Files müssen einmal vor dem eigentlichen Programmstart von Diskette geladen werden. Wem diese Zweiteilung des Programms nicht gefällt, kann die Bildschirme auch direkt ins Programm einbinden. Dazu muß beim Abspeichern der Texte die Startadresse 3291 angegeben werden. Der Deconverter lädt das File dann direkt an seinen Basicanfang. Mit dem Befehl <BASIC> wird der Basicanfang hinter die Texte

gelegt. Um ein jetzt eingetipptes oder eingeladenes Programm zusammen mit dem Deconverter und den Texten abspeichern zu können, muß nur der Befehl <NRM> eingegeben werden.

Die Möglichkeit der Speicherverschiebung hat einen weiteren Vorteil: Es läßt sich problemlos Platz für Sprites, Grafiken oder Maschinenprogramme schaffen, die dann zusammen mit dem TCS abgespeichert werden können. Zu diesem Zweck kann man den Basicanfang auch an eine beliebige Adresse verschieben.

4. Geschwindigkeit

Ein großer Vorteil des TCS ist, daß sich Texte wesentlich schneller auf den Bildschirm bringen lassen als mit dem PRINT-Befehl. In der Tabelle sind einige Meßwerte zusammengestellt. Die Zahlen in Klammern geben die Zeiten ohne Bildschirmlöschen an.

„Komplett-BS" ist der komplexeste Bildschirm, der überhaupt möglich ist. Der Bildschirm ist komplett mit Zeichen gefüllt, die abwechselnd invertiert sind und verschiedene Farben haben. Das ist natürlich ein rein theoretischer Fall, der aber die Leistungsfähigkeit des TCS gut verdeutlicht: Die Geschwindigkeitssteigerung beträgt knapp 250%!

„Voller BS" ist eine aufwendige Bildschirmmaske mit vielen Revers- und Farbumschaltungen. Auch hier ist das TCS 3x schneller.

„BS 30%" ist ein einfaches Auswahlmenü. Das TCS braucht zum Bildschirmlöschen generell wesentlich weniger Zeit als der PRINT-Befehl.

Bei nur einer Zeile fällt der Geschwindigkeitsvorteil kaum noch ins Gewicht.

Testzeiten in s bei 100 Durchläufen

	PRINT		TCS	
Komplett-BS	132,6	(128,5)	38,5	(38,5)
Voller BS	51,7	(47,7)	15,9	(15,5)
BS 30%	22,1	(18,5)	6,6	(4,5)
1 Zeile	5,9	(1,8)	3,9	(1,3)

Einsatzbereich

Beim TCS handelt es sich um ein recht ungewöhnliches Utility. Deshalb sollen hier einige Vor- und Nachteile des Programms aufgelistet werden. Dadurch kristallisiert sich auch der Anwendungsbereich des TCS deutlich heraus.

Vorteile

— Die 20 KByte RAM unter dem ROM können benutzt werden
  = > Programmlistings werden kürzer und übersichtlicher

— Bildschirme werden gepackt
  = > Speicherplatzersparnis bei größeren Programmen

— Die Bildschirme werden wesentlich schneller ausgedruckt
  = > Professionelle Programmgestaltung in Basic

— Verschiebung des Basicspeichers einfach möglich
  = > Einfache Einbindung von MC-Routinen und Grafiken

— Es können beliebig viele Textfiles nachgeladen werden
  = > Ein Basicprogramm kann sehr große Textmengen verwalten

— Auch von Maschinensprache aus nutzbar

— Listschutz

Nachteile

— Die Bildschirme können nach dem Packen nicht mehr verändert werden

— Relativ aufwendige Handhabung

— Die Texte müssen nachgeladen werden
  (Läßt sich umgehen)

Wägt man diese Argumente ab, so zeigt sich schnell, wo sich das TCS sinnvoll einsetzen läßt. Die größte Effektivität erreicht man bei längeren Programmen (oder solche mit großen Datenfeldern), die viele Bildschirmmasken enthalten. Hier kann man in der Endphase der Programmierung durch Packen der Texte wertvollen Speicherplatz sparen und das Programm vereinfachen.

Mit etwas Phantasie lassen sich bestimmt noch eine Menge weiterer Einsatzgebiete für das TCS finden. Wie wäre es zum Beispiel mit einer Diashow aus Textbildschirmen?

Auf der beiliegenden Diskette sind noch zwei kurze Beispielprogramme gespeichert, die einen kleinen Einblick in die Praxis der Programmierung mit dem TCS geben.

4/6.3.9.2

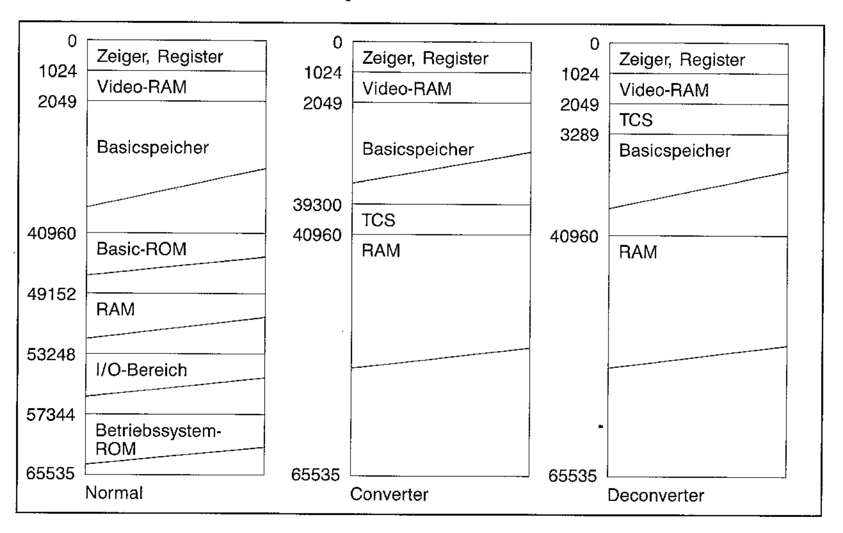
Programmdokumentation

Speicherbelegung

Die beiden Teilprogramme des TCS belegen unterschiedliche Speicherbereiche, wie in den Grafiken dargestellt.

Der Converter
liegt am Ende des Basicspeichers, das Basicende wird beim Start heruntergesetzt. Der restliche Basicspeicher sowie der gesamte ROM-Bereich ab 40960 stehen für Texte zur Verfügung.

Der Deconverter
liegt am Anfang des Basicspeichers, der Basicanfang wird direkt an das Programmende verschoben. Jetzt folgende Programme können leicht zusammen mit der Befehlserweiterung abgespeichert werden. Außerdem liegt hinter dem Converter dadurch lückenlos RAM bis zum Speicherende.



Nutzung in Assembler

Der folgende Abschnitt beschäftigt sich mit dem Einsatz des Deconverters in Maschinensprache. In der folgenden Aufstellung wird zuerst die Startadresse der Routine und dann die Einsprungadresse aus einem Assemblerprogramm heraus angegeben. Alle Routinen enden mit einem RTS. Bei Parametereinstellungen werden statt dem Einsprung die entsprechenden Speicherstellen angegeben.

Routine	Startadresse	Einsprungadresse	Parameter
NRM	$0bae	$0bae	keine
PRG	$0bb7	$0bb7	keine
DISK	$0abb	$0ac7	Der PRG-Name muß gesetzt sein (SETFLS).
BS	$0880	$0883	Bildschirmnummer ins X-Register.
SCREEN	$0bc2 Bildschirmanfang in die Adressen $0851/$0852		
NUM	$0bf5	$0bf5	keine
MEM	$0bcc	$0bcc	keine
SPACE	$0bee Farbe der Leerzeichen in Adresse $0842		
BASIC	$0b6d	ohne Parameter: $0b72 mit Parameter: $0b86, 10/lo des Anfangs in A/Y	
OFF	$0832	$0832	keine

Pack-Algorithmus

Beim Packen der Bildschirme benutzt das TCS einen speziellen Algorithmus, der den Bildschirminhalt komprimiert. Es soll nicht der Versuch gemacht werden, diesen doch recht komplexen Programmteil anhand des Listings zu erklären (Interessierte seien auf den vollständig dokumentierten Quelltext verwiesen). Vielmehr soll das Prinzip erläutert werden, um Stärken und Schwächen des Programms herauszustellen.

Nach dem Aufruf des Converters und der Bestätigung mit <RETURN> wird zuerst die Startadresse des folgenden Bildschirms in einer Tabelle abgelegt. Falls nötig, wird die Hintergrund-/Rahmenfarbe gespeichert.

Dann beginnt der eigentliche Packvorgang. Der Bildschirm wird zeilenweise abgetastet und die Zeichen im Bildschirmcode im Speicher abgelegt. Bei Farbwechseln und reversen Zeichen werden genau wie beim PRINT-Befehl Steuerzeichen vorangestellt (siehe Tabelle).

Stehen am Zeilenende mehrere Leerzeichen, so weden diese durch ein Sonderzeichen ersetzt.

Wiederholt sich das gleiche Zeichen mehrmals hintereinander, so wird die Zeichenfolge auf drei Bytes gekürzt (Signalcode, Zeichen, Anzahl).

Treten auf dem Bildschirm mehrere Leerzeilen hintereinander auf, so werden diese in zwei Bytes im Speicher vermerkt (Signalcode, Anzahl).

Funktion der Steuerzeichen

Steuerzeichen Code	Funktion
128	Revers ein
129	Revers aus
130–145	Zeichenfarbe
146	Zeilenende
147	gleiche Zeichen
148–173	Leerzeilen

Am Ende des Packvorganges wird die laufende Speicheradresse gemerkt und zurück zum Basic gesprungen.

4/6.3.9.3

Übersichten

Programmteil 1: Converter	Programmteil 2: Deconverter
Name : TCS Converter Anfang : $9984 (39300) Ende : $9ff0 (40944) Länge : $066c (1644) Gesamtlänge: 2886 Bytes	Name : TCS Deconverter Anfang : $0801 (2049) Ende : $0cdb (3291) Länge : $04da (1242)

Verwendete Speicherstellen in der Zeropage:

$f7–$fc Kann auch von anderen Programmen verwendet werden.
$02 Flag, das nicht überschrieben werden sollte.

Directory:

```
0    "T.C.S.                    " A1 2A
0      "------------------------" DEL
0      "TEXT                    " DEL
0      "        CONVERT         " DEL
0      "                  SYSTEM" DEL
0      "------------------------" DEL
0      "                        " DEL
7      "TCS CONVERTER"            PRG
5      "TCS DECONVERTER"          PRG
0      "------------------------" DEL
7      "TCS DEMO 1"               PRG
4      "TCS DEMO 1.BS"            PRG
7      "TCS DEMO 2"               PRG
7      "TCS DEMO 2.BS"            PRG
0      "------------------------" DEL
71     "CONVERTER.QT"             PRG
56     "DECONVERTER.QT"           PRG
500 BLOCKS FREE.

READY.
```

Befehlsübersicht

Hier eine Übersicht der neuen Befehle, die beide Programmteile zur Verfügung stellen.

Converter

INIT	Tabelle anlegen und löschen, dann TAST 1
START	Legt Anfang der Texte fest, danach INIT
TAST	Tastaturvektor wird ein-/ausgeschaltet
SCREEN	Legt den Anfang des Bildschirmspeichers fest
COL	Bildschirmfarben sichern ja/nein
MEM	Gibt den aktuellen Speicherstand aus
NUM	Anzahl der bisher gepackten Bildschirme
DISK	Speichert die Bildschirme ab
OFF	Schaltet die Erweiterung aus

Deconverter

NRM	Normaler Anfang des Basicspeichers
PRG	Basicanfang auf den Programmanfang
DISK	Lädt ein Textfile
BS	Gibt einen Bildschirm aus
SCREEN	Legt den Anfang des Bildschirmspeichers fest
NUM	Anzahl der im Speicher liegenden Bildschirme
MEM	Gibt Anfang- und Endadresse der Texte an
SPACE	Farbe der Leerzeichen, Window-Modus
BASIC	Legt den Basicanfang fest
OFF	Schaltet die Erweiterung aus

Fehlermeldungen

SCREEN ERROR	Der Deconverter ist beim Auspacken eines Bildschirms auf undefinierte Zeichen gestoßen.
NO TCS SCREEN ERROR	Es wurde versucht, mit dem DISK-Befehl ein File einzuladen, das nicht vom TCS stammt.
NO SCREEN DEFINED	Es wurde versucht, Textfiles abzuspeichern, obwohl keine Bildschirme gepackt worden sind.

```
0 IFPEEK(820)<>255THENPOKE820,255:DISK"TCS DEMO 1.BS"
5 POKE650,128:PRINTCHR$(14)
10 SPACE0:BS1: REM MENUE
20 POKE198,0:WAIT198,1:GETA$:SPACE16
30 IFA$="(F1)"THEN:BS2:GOSUB100:GOTO10
40 IFA$="(F3)"THEN:BS2:GOSUB100:BS3:GOSUB100:SPACE0:BS1:SPACE
16:BS2:GOSUB100:GOTO10
50 IFA$="(F5)"THEN:BS2:GOSUB100:BS4:GOSUB100:SPACE 0:BS5:GOSU
B100:GOTO10
60 IFA$="(F7)"THENEND
70 GOTO10
100 POKE198,0:WAIT198,1:RETURN
```

Basic-Listing 4/6.3.9.3-1

```
5 IFPEEK(820)<>99THENPOKE820,99:DISK"TCS DEMO 2.BS"
10 PRINTCHR$(142):SPACE0:BS1:SPACE16
20 T=2:W=400
30 BS T:T=T+1:IFT=6THENT=2
40 FORA=1TOW:NEXT
50 IFF=0THENW=W-(W/10):IFW<1.1THENF=1
60 IFF=1THENW=W+(W/10):IFW>400THENF=0
70 GOTO30
```

Basic-Listing 4/6.3.9.3-2

```
hypra-ass  assemblerlisting:

             10   -.li1,4,7
             20   -.ba $c801; normal: $0801 (basic-anfang)
             30   -.wo nz
             40   -.wo 1988
             50   -.by $9e
             60   -.tx"2084: "
             70   -.by$8f
             80   -.tx" text-convert-system"
             90   -.by 00
             100  -nz        .wo 00
;
;
;******************************
;**                          **
;**  TEXT - CONVERT - SYSTEM **
;**                          **
;**   Quelltext Deconverter  **
;**                          **
;******************************
;**                          **
;******************************
;
;
;------ variablen extern --------
;
             320  -.eq bs    = $f7
             330  -.eq sp    = $f9
             340  -.eq cl    = $fb
             350  -.eq start = $02
;
;--------- konstanten -----------
;
             390  -.eq revon    = 128
             400  -.eq revoff   = 129
             410  -.eq space    =  32
             420  -.eq color    = 130
             430  -.eq linend   = 146
;
;--------- routinen -------------
;
             470  -.eq get      = $ffe4
             480  -.eq open     = $ffc0
             490  -.eq setfls   = $ffba
             500  -.eq setnam   = $ffbd
             510  -.eq close    = $ffc3
             520  -.eq chkin    = $ffc6
             530  -.eq ckout    = $ffc9
             540  -.eq clrch    = $ffcc
             550  -.eq bsout    = $ffd2
```

Listing 4/6.3.9.3-1 (Teil 1)

```
                 560   -.eq chkcom    = $aefd
                 570   -.eq getbyte   = $b79e
                 580   -.eq frmnum    = $ad8a
                 590   -.eq integer   = $b7f7
                 600   -.eq clr       = $e544
                 610   -.eq illquan   = $b248
                 620   -.eq errout    = $a445
                 630   -.eq chrgot    = $  79
                 640   -.eq linprt    = $bdcd
                 650   -.eq intvec    = $0308
                 660   -.eq chrget    = $73
                 670   -.eq chrgot    = $79
                 680   -.eq intend    = $a7e7
                 690   -.eq interalt  = $a7e4
                 700   -.eq txtptr    = $7a
;
;
;---------------------------------
;--     befehlsvector umbiegen   --
;---------------------------------
;
c824 4cb2cc :770   -            jmp extend
c827 a908   :780   -interon     lda #<(inter)   ;befehls-
c829 a0cc   :790   -            ldy #>(inter)   ;vector
c82b 8d0803 :800   -            sta intvec      ;auf neue
c82e 8c0903 :810   -            sty intvec+1    ;routine
c831 60     :820   -            rts             ;ende
;
c832 a9e4   :840   -interoff    lda #<(interalt);bef.-
c834 a0a7   :850   -            ldy #>(interalt);vector
c836 8d0803 :860   -            sta intvec      ;zurueck-
c839 8c0903 :870   -            sty intvec+1    ;stellen
c83c 60     :880   -            rts             ;ende
;
;---------------------------------
;--     variablen prg-intern     --
;---------------------------------
;
                 940   -code        .by 00
                 950   -code2       .by 00
                 960   -revflag     .by 00
                 970   -line        .by 00
                 980   -col         .by 00
                 990   -spcol       .by 00
                 1000  -yreg        .by 00
                 1010  -count       .by 00
                 1020  -flag1       .by 00
                 1030  -konf        .by 00
                 1040  -akku        .by 00
                 1050  -anf1        .by 00
                 1060  -anf2        .by 00
```

Listing 4/6.3.9.3-1 (Teil 2)

```
                1070 -end1       .by 00
                1080 -end2       .by 00
                1090 -anf3       .by 00
                1100 -anf4       .by 00
                1110 -spend      .wo basicanf+1
                1120 -anzahl     .by 00
                1130 -bsconst    .wo $0400
                1140 -clconst    .wo $d800
                1150 -prg        .wo basicanf+1
                1160 -flag2      .by 00
                1170 -errmsg     .tx "screeN"
                1180 -errmsg1    .tx "no tcs screeN"
                1190 -befnr      .by 00
                1200 -window     .by 00
;
;
;--------------------------------
;--    basicanf. verschieben   --
;--------------------------------
;
c86d 2027c8 :1270 -init        jsr interon
c870 ad55c8 :1280 -            lda prg          ;basic
c873 852b   :1290 -            sta 43           ;speicher
c875 ad56c8 :1300 -            lda prg+1        ;ver-
c878 852c   :1310 -            sta 44           ;schieben
c87a 2059a6 :1320 -            jsr $a659
c87d 4caea7 :1330 -            jmp $a7ae        ;RUN
;
;--------------------------------
;--        deconverter         --
;--------------------------------
;
c880 209eb7 :1390 -deconvert   jsr getbyte      ;bs-nr.
c883 e000   :1400 -            cpx #0           ;=0 ?
c885 f006   :1410 -            beq illq
c887 ca     :1420 -            dex              ;hoeher
c888 ec50c8 :1430 -            cpx anzahl       ;als er-
c88b 9003   :1440 -            bcc decon2       ;laubt ?
c88d 4c48b2 :1450 -illq        jmp illquan
;
c890 86f9   :1470 -decon2      stx sp           ;speichern
c892 a900   :1480 -            lda #0           ;high=0
c894 85fa   :1490 -            sta sp+1
;
c896 18     :1510 -            clc
c897 06f9   :1520 -            asl sp           ;mal 2
c899 26fa   :1530 -            rol sp+1
;
c89b 18     :1550 -            clc              ;tabellen-
c89c ad48c8 :1560 -            lda anf1         ;anfang
```

Listing 4/6.3.9.3-1 (Teil 3)

```
c89f 65f9    :1570 -          adc sp          ;addieren
c8a1 85f9    :1580 -          sta sp
c8a3 ad49c8  :1590 -          lda anf2
c8a6 65fa    :1600 -          adc sp+1
c8a8 85fa    :1610 -          sta sp+1
;
c8aa a000    :1630 -          ldy #0          ;tabelle
c8ac 2087ca  :1640 -          jsr read        ;auslesen
c8af 8d47c8  :1650 -          sta akku
c8b2 2089c9  :1660 -          jsr spinc
c8b5 2087ca  :1670 -          jsr read
c8b8 85fa    :1680 -          sta sp+1        ;text-
c8ba ad47c8  :1690 -          lda akku        ;anfang
c8bd 85f9    :1700 -          sta sp          ;speichern
;
c8bf ad51c8  :1720 -          lda bsconst     ;bs-anf.
c8c2 ac52c8  :1730 -          ldy bsconst+1
c8c5 85f7    :1740 -          sta bs
c8c7 84f8    :1750 -          sty bs+1
;
c8c9 ad53c8  :1770 -          lda clconst     :farb-
c8cc ac54c8  :1780 -          ldy clconst+1   ;speicher
c8cf 85fb    :1790 -          sta cl
c8d1 84fc    :1800 -          sty cl+1
;
;
c8d3 a900    :1830 -          lda #0          ;zeiger
c8d5 8d3fc8  :1840 -          sta revflag     :loeschen
c8d8 8d41c8  :1850 -          sta col
c8db 8d40c8  :1860 -          sta line
c8de 8d43c8  :1870 -          sta yreg
;
;--------- bs farben ------------
;
c8e1 2087ca  :1910 -          jsr read        ;bs-farben
c8e4 c9ff    :1920 -          cmp #255        ;setzen ?
c8e6 d016    :1930 -          bne loop2x
;
c8e8 18      :1950 -          clc             ;rahmen-
c8e9 2089c9  :1960 -          jsr spinc       ;farbe
c8ec 2087ca  :1970 -          jsr read
c8ef 8d20d0  :1980 -          sta $d020
c8f2 2089c9  :1990 -          jsr spinc       ;bs-farbe
c8f5 2087ca  :2000 -          jsr read
c8f8 8d21d0  :2010 -          sta $d021
c8fb 2089c9  :2020 -          jsr spinc
;
;--------- hauptschleife 2-------
;
c8fe a000    :2060 -loop2x    ldy #00
c900 8c6cc8  :2070 -          sty window
```

Listing 4/6.3.9.3-1 (Teil 4)

```
c903 8c43c8  :2080 -loop1x    sty yreg       ;y sichern
c906 a000    :2090 -          ldy #0
c908 18      :2100 -          clc
c909 2087ca  :2110 -          jsr read
c90c 8d3dc8  :2120 -          sta code
c90f ac43c8  :2130 -          ldy yreg
;
c912 c980    :2150 -          cmp #revon     ;steuer-
c914 b07b    :2160 -          bcs steuerz    ;zeichen ?
;
c916 ad3fc8  :2180 -          lda revflag    ;reverses
c919 f008    :2190 -          beq speicherx  ;zeichen ?
c91b ad3dc8  :2200 -          lda code
c91e 6980    :2210 -          adc #revon
c920 8d3dc8  :2220 -          sta code
;
c923 ad3dc8  :2240 -speicherx lda code       ;leer-
c926 c920    :2250 -          cmp #space     ;zeichen ?
c928 f040    :2260 -          beq spaceb     ;ja
;
c92a ad3dc8  :2280 -          lda code       ;zeichen
c92d 91f7    :2290 -          sta (bs),y     ;in bilds.
;
c92f ad41c8  :2310 -          lda col        ;farbe
c932 91fb    :2320 -          sta (cl),y
c934 ee6cc8  :2330 -          inc window
;
c937 2089c9  :2350 -zaehlerx  jsr spinc      ;sp+1
c93a c8      :2360 -          iny
c93b c028    :2370 -          cpy #40        ;zeilen-
c93d d0c4    :2380 -          bne loop1x     ;ende ?
;
c93f a5f7    :2400 -newlinex  lda bs         ;naechste
c941 18      :2410 -          clc            ;zeile
c942 6928    :2420 -          adc #40        ;bild-
c944 85f7    :2430 -          sta bs         ;schirmsp.
c946 a5f8    :2440 -          lda bs+1
c948 6900    :2450 -          adc #0
c94a 85f8    :2460 -          sta bs+1
;
c94c a5fb    :2480 -          lda cl         ;naechste
c94e 18      :2490 -          clc            ;zeile
c94f 6928    :2500 -          adc #40        ;farbsp.
c951 85fb    :2510 -          sta cl
c953 a5fc    :2520 -          lda cl+1
c955 6900    :2530 -          adc #0
c957 85fc    :2540 -          sta cl+1
;
c959 ad45c8  :2560 -          lda flag1
c95c f001    :2570 -          beq newlinex1
```

Listing 4/6.3.9.3-1 (Teil 5)

```
c95e 60      :2580 -            rts
;
;
c95f ee40c8 :2610 -newlinex1 inc line       ;letzte
c962 ad40c8 :2620 -           lda line       ;zeile ?
c965 c919    :2630 -           cmp #25
c967 d095    :2640 -           bne loop2x
;
c969 60      :2660 -ende2      rts
;
c96a ad42c8 :2680 -spaceb     lda spcol      ;bildsch.
c96d c910    :2690 -           cmp #16        ;loeschen?
c96f b00c    :2700 -           bcs space2     ;nein
;
c971 ad42c8 :2720 -space3     lda spcol      ;zeichen
c974 91fb    :2730 -           sta (cl),y     ;loeschen
c976 a920    :2740 -           lda #space     ;und farbe
c978 91f7    :2750 -           sta (bs),y     ;setzen
c97a 4c37c9 :2760 -           jmp zaehlerx
;
c97d c910    :2780 -space2     cmp #16        ;window-
c97f d0b6    :2790 -           bne zaehlerx   ;modus?
c981 ad6cc8 :2800 -           lda window     ;w=0 nicht
c984 f0b1    :2810 -           beq zaehlerx   ;loeschen
c986 4c71c9 :2820 -           jmp space3     ;clear
;
;------------ spinc -------------
;
c989 e6f9    :2860 -spinc      inc sp         ;speicher
c98b d002    :2870 -           bne spinc1     ;zaehler
c98d e6fa    :2880 -           inc sp+1       ;+1
c98f 18      :2890 -spinc1     clc
c990 60      :2900 -           rts
;
;-------- steuerzeichen -----------
;
c991 c980    :2940 -steuerz    cmp #revon     ;revon
c993 d00b    :2950 -           bne steuerz2   ;flag ?
c995 a901    :2960 -           lda #1
c997 8d3fc8 :2970 -           sta revflag
c99a 2089c9 :2980 -           jsr spinc
c99d 4c03c9 :2990 -           jmp loop1x
;
c9a0 c981    :3010 -steuerz2   cmp #revoff    ;revoff
c9a2 d00b    :3020 -           bne steuerz3   ;flag
c9a4 a900    :3030 -           lda #0
c9a6 8d3fc8 :3040 -           sta revflag
c9a9 2089c9 :3050 -           jsr spinc
c9ac 4c03c9 :3060 -           jmp loop1x
;
c9af c992    :3080 -steuerz3   cmp #linend    ;zeilen-
```

Listing 4/6.3.9.3-1 (Teil 6)

```
c9b1 b00d   :3090 -          bcs zende        ;ende ?
c9b3 38     :3100 -          sec
c9b4 e982   :3110 -          sbc #color       ;neue
c9b6 8d41c8 :3120 -          sta col          ;farbe
c9b9 18     :3130 -          clc
c9ba 2089c9 :3140 -          jsr spinc
c9bd 4c03c9 :3150 -          jmp loop1x
;
;------- naechste zeile ---------
;
c9c0 c993   :3190 -zende     cmp #147         ;code fuer
c9c2 b027   :3200 -          bcs doub         ;leerzeile
;
c9c4 ad42c8 :3220 -zende3    lda spcol        ;bs nicht
c9c7 c910   :3230 -          cmp #16          ;loeschen?
c9c9 b014   :3240 -          bcs zende2
;
c9cb a920   :3260 -zende1    lda #space       ;bs-stelle
c9cd 91f7   :3270 -          sta (bs),y       ;loeschen
;
c9cf ad42c8 :3290 -          lda spcol        ;letzte
c9d2 91fb   :3300 -          sta (cl),y       ;farbe
;
c9d4 c8     :3320 -          iny              ;bis zei-
c9d5 c028   :3330 -          cpy #40          ;lenende
c9d7 d0f2   :3340 -          bne zende1       ;naechste
c9d9 ad45c8 :3350 -          lda flag1
c9dc f001   :3360 -          beq zende2
c9de 60     :3370 -          rts
c9df ad45c8 :3380 -zende2    lda flag1
c9e2 f001   :3390 -          beq zende4
c9e4 60     :3400 -          rts
c9e5 2089c9 :3410 -zende4    jsr spinc
c9e8 4c3fc9 :3420 -          jmp newlinex     ;zeile
;
;------ gleiche zeichen ---------
;
c9eb c994   :3460 -doub      cmp #148
c9ed b058   :3470 -          bcs scrend
;
c9ef 8c43c8 :3490 -          sty yreg         ;y sichern
c9f2 a000   :3500 -          ldy #0
c9f4 2089c9 :3510 -          jsr spinc        ;anzahl
c9f7 2087ca :3520 -          jsr read         ;holen
c9fa 8d44c8 :3530 -          sta count
c9fd 2089c9 :3540 -          jsr spinc        ;zeichen
;
ca00 2087ca :3560 -          jsr read         ;zeichen
ca03 8d3ec8 :3570 -          sta code2        ;holen
ca06 ac43c8 :3580 -          ldy yreg         ;y holen
;
```

Listing 4/6.3.9.3-1 (Teil 7)

```
ca09 ad3ec8 :3600 -doub2     lda code2      ;leer-
ca0c c920   :3610 -          cmp #space     ;zeichen ?
ca0e f018   :3620 -          beq spacec
;
ca10 ad3ec8 :3640 -          lda code2
ca13 91f7   :3650 -          sta (bs),y     ;in bs
ca15 ad41c8 :3660 -          lda col        ;farbe
ca18 91fb   :3670 -          sta (cl),y
ca1a 18     :3680 -doub3     clc
;
ca1b c8     :3700 -          iny            ;bs+1
ca1c ce44c8 :3710 -          dec count      ;anz-1
ca1f ad44c8 :3720 -          lda count      ;letztes
ca22 d0e5   :3730 -          bne doub2      ;zeichen ?
ca24 88     :3740 -          dey
ca25 4c37c9 :3750 -          jmp zaehlerx   ;weiter
;
ca28 ad42c8 :3770 -spacec    lda spcol      ;bs
ca2b c911   :3780 -          cmp #17        ;loeschen?
ca2d b00d   :3790 -          bcs spacec2
ca2f c910   :3800 -          cmp #16        ;window-
ca31 f00c   :3810 -          beq spacec3    ;modus ?
;
ca33 ad42c8 :3830 -spacec1   lda spcol
ca36 91fb   :3840 -          sta (cl),y     ;farbe &
ca38 a920   :3850 -          lda #space     ;zeichen
ca3a 91f7   :3860 -          sta (bs),y     ;setzen
ca3c 4c1aca :3870 -spacec2   jmp doub3      ;zurueck
;
ca3f ad6cc8 :3890 -spacec3   lda window     ;loeschen?
ca42 d0ef   :3900 -          bne spacec1    ;ja
ca44 4c1aca :3910 -          jmp doub3      ;nein
;
;---- mehrere leerzeilen --------
;
ca47 38     :3950 -scrend    sec            ;code
ca48 e994   :3960 -          sbc #148       ;abziehen
ca4a 18     :3970 -          clc
ca4b 8d44c8 :3980 -          sta count
;
ca4e c91a   :4000 -          cmp #26        ;fremder
ca50 9009   :4010 -          bcc scrend3    ;code ?
;
ca52 a958   :4030 -          lda #<(errmsg) ;ja,
ca54 8522   :4040 -          sta $22        ;fehler-
ca56 a9c8   :4050 -          lda #>(errmsg) ;meldung
ca58 4c45a4 :4060 -          jmp errout     ;ausgeben
;
ca5b 2089c9 :4080 -scrend3   jsr spinc
ca5e a901   :4090 -          lda #1         ;rest der
ca60 8d45c8 :4100 -          sta flag1      ;zeile
```

Listing 4/6.3.9.3-1 (Teil 8)

```
ca63 20c4c9 :4110 -            jsr zende3     ;loeschen
ca66 ee40c8 :4120 -            inc line       ;zeiger+1
;
ca69 a000   :4140 -scrend1     ldy #0         ;zeiger
ca6b 203fc9 :4150 -            jsr newlinex   ;+1
ca6e 20c4c9 :4160 -            jsr zende3     :zeile
ca71 ce44c8 :4170 -            dec count      ;loeschen
ca74 ee40c8 :4180 -            inc line
ca77 ad44c8 :4190 -            lda count      ;letzte
ca7a d0ed   :4200 -            bne scrend1    ;zeile ?
;
ca7c a900   :4220 -            lda #0         ;flag =0
ca7e 8d45c8 :4230 -            sta flag1      ;und
ca81 ce40c8 :4240 -            dec line       ;weiter
ca84 4c3fc9 :4250 -scrend2     jmp newlinex
;
;------ speicher auslesen -------
;
ca87 a501   :4290 -read        lda 1          ;speicher
ca89 8d46c8 :4300 -            sta konf       ;auf ram
ca8c 29f8   :4310 -            and #248       ;schalten
ca8e 78     :4320 -            sei
ca8f 8501   :4330 -            sta 1          ;wert
ca91 a000   :4340 -            ldy #0
ca93 b1f9   :4350 -            lda (sp),y     ;holen
ca95 ac46c8 :4360 -            ldy konf
ca98 8401   :4370 -            sty 1          ;speicher
ca9a 58     :4380 -            cli            ;zurueck
ca9b a000   :4390 -            ldy #0
ca9d 60     :4400 -            rts
;
ca9e 8d47c8 :4420 -write       sta akku
caa1 a501   :4430 -            lda 1          ;speicher
caa3 8d46c8 :4440 -            sta konf       ;auf ram
caa6 29f8   :4450 -            and #248       ;schalten
caa8 78     :4460 -            sei
caa9 8501   :4470 -            sta 1
caab a000   :4480 -            ldy #0
caad ad47c8 :4490 -            lda akku       ;wert
cab0 91f9   :4500 -            sta (sp),y     ;speicher
cab2 ac46c8 :4510 -            ldy konf
cab5 8401   :4520 -            sty 1          ;speicher
cab7 58     :4530 -            cli            ;zurueck
cab8 a000   :4540 -            ldy #0
caba 60     :4550 -            rts
;
;--------------------------------
;--          datei laden      --
;--------------------------------
;
cabb 2057e2 :4610 -load        jsr $e257      ;name
```

Listing 4/6.3.9.3-1 (Teil 9)

```
;
cabe a902    :4630 -load5     lda #2         ;filenr.
cac0 a208    :4640 -          ldx #8         ;8=floppy
cac2 a002    :4650 -          ldy #2         ;sek.adr.
cac4 20baff  :4660 -          jsr setfls
;
cac7 20c0ff  :4680 -load7     jsr open       ;file
caca a590    :4690 -          lda $90
cacc f003    :4700 -          beq load9
cace 4c58cb  :4710 -          jmp loadend
cad1 a202    :4720 -load9     ldx #2         ;oeffnen
cad3 20c6ff  :4730 -          jsr chkin
;
cad6 a590    :4750 -          lda $90
cad8 d07e    :4760 -          bne loadend
;
cada 20e4ff  :4780 -          jsr get        ;auf die
cadd c954    :4790 -          cmp #"t"       ;kennung
cadf f000    :4800 -          beq load12     ;'tcs'
cae1 a590    :4810 -load12    lda $90        ;pruefen
cae3 d073    :4820 -          bne loadend    ;falls
cae5 20e4ff  :4830 -          jsr get        ;nicht,
cae8 c943    :4840 -          cmp #"c"       ;routine
caea d075    :4850 -          bne load11     ;beenden
caec 20e4ff  :4860 -          jsr get
caef c953    :4870 -          cmp #"s"
caf1 d06e    :4880 -          bne load11
;
caf3 20e4ff  :4900 -load6     jsr get        ;anfangs-
caf6 85f9    :4910 -          sta sp         ;adresse
caf8 8d48c8  :4920 -          sta anf1
cafb a590    :4930 -          lda $90
cafd d059    :4940 -          bne loadend
caff 20e4ff  :4950 -          jsr get        ;holen
cb02 85fa    :4960 -          sta sp+1
cb04 8d49c8  :4970 -          sta anf2
;
cb07 20e4ff  :4990 -          jsr get        ;anzahl
cb0a 8d50c8  :5000 -          sta anzahl     ;bs holen
;
cb0d 20e4ff  :5020 -load2     jsr get        ;tabelle
cb10 209eca  :5030 -          jsr write      ;auslesen
cb13 2089c9  :5040 -          jsr spinc      ;und
cb16 c900    :5050 -          cmp #0         ;speichern
cb18 d0f3    :5060 -          bne load2      ;byte=0 ?
;
cb1a 20e4ff  :5080 -          jsr get        ;tabellen
cb1d 209eca  :5090 -          jsr write      ;ende ?
cb20 2089c9  :5100 -          jsr spinc
cb23 c900    :5110 -          cmp #0
cb25 d0e6    :5120 -          bne load2
```

Listing 4/6.3.9.3-1 (Teil 10)

```
;
cb27 a5f9    :5140 -           lda sp          ;anf.
cb29 a4fa    :5150 -           ldy sp+1        ;textsp.
cb2b 8d4cc8  :5160 -           sta anf3        ;merken
cb2e 8c4dc8  :5170 -           sty anf4
;
cb31 20e4ff  :5190 -load1      jsr get         ;zeichen
cb34 a000    :5200 -           ldy #0          ;holen &
cb36 209eca  :5210 -           jsr write       ;speichern
;
cb39 a590    :5230 -           lda $90         ;letztes
cb3b d006    :5240 -           bne load8       ;zeichen ?
cb3d 2089c9  :5250 -           jsr spinc       ;nein,
cb40 4c31cb  :5260 -           jmp load1       ;weiter
;
cb43 2089c9  :5280 -load8      jsr spinc       ;naechstes
cb46 a900    :5290 -           lda #0          ;byte =0
cb48 209eca  :5300 -           jsr write
cb4b 2089c9  :5310 -           jsr spinc       ;zaehler+1
cb4e a5f9    :5320 -           lda sp          ;als
cb50 a4fa    :5330 -           ldy sp+1        ;text-ende
cb52 8d4ec8  :5340 -           sta spend       ;ab-
cb55 8c4fc8  :5350 -           sty spend+1     ;speichern
;
cb58 20ccff  :5370 -loadend    jsr clrch       ;datei
cb5b a902    :5380 -           lda #2          ;schliess-
cb5d 20c3ff  :5390 -           jsr close       ;en
cb60 60      :5400 -           rts             ;ende
;
cb61 2058cb  :5420 -load11     jsr loadend
cb64 a95e    :5430 -           lda #<(errmsg1)
cb66 8522    :5440 -           sta $22
cb68 a9c8    :5450 -           lda #>(errmsg1)
cb6a 4c45a4  :5460 -           jmp errout
;
;--------------------------------
;-- parameter aendern/anzeigen --
;--------------------------------
;
;---- basicanfang verschieben ---
;
cb6d 207900  :5540 -prganf     jsr chrgot      ;folgen
cb70 d011    :5550 -           bne prganf1     ;param. ?
;
cb72 ad4ec8  :5570 -           lda spend       ;textende
cb75 ac4fc8  :5580 -           ldy spend+1     ;als
cb78 852b    :5590 -           sta 43          ;basic-
cb7a 8d55c8  :5600 -           sta prg         ;anfang
cb7d 842c    :5610 -           sty 44
cb7f 8c56c8  :5620 -           sty prg+1
cb82 60      :5630 -           rts             ;basic
```

Listing 4/6.3.9.3-1 (Teil 11)

```
;
cb83 20a3cb :5650 -prganf1    jsr parameter+3;basic-
cb86 8d55c8 :5660 -           sta prg         ;anfang
cb89 852b   :5670 -           sta 43          ;holen &
cb8b 8c56c8 :5680 -           sty prg+1       ;speichern
cb8e 842c   :5690 -           sty 44
;
cb90 38     :5710 -           sec             ;vom basic
cb91 e901   :5720 -           sbc #1          ;anfang
cb93 85f9   :5730 -           sta sp          ;eins ab-
cb95 98     :5740 -           tya             ;ziehen
cb96 e900   :5750 -           sbc #0
cb98 85fa   :5760 -           sta sp+1
;
cb9a a000   :5780 -           ldy #0          ;speicher-
cb9c 98     :5790 -           tya             ;stelle
cb9d 91f9   :5800 -           sta (sp),y      ;auf null
cb9f 60     :5810 -           rts             ;basic
;
;
cba0 20fdae :5840 -parameter jsr chkcom      ;komma ?
cba3 208aad :5850 -           jsr frmnum      ;zahl
cba6 20f7b7 :5860 -           jsr integer     ;holen
cba9 a514   :5870 -           lda $14
cbab a415   :5880 -           ldy $15
cbad 60     :5890 -           rts             ;zurueck
;
;---- basicspeicher anfang -----
;
cbae a901   :5930 -norm       lda #1          ;basicanf.
cbb0 a008   :5940 -           ldy #8          ;auf 2048
cbb2 852b   :5950 -           sta 43          ;sezten
cbb4 842c   :5960 -           sty 44
cbb6 60     :5970 -           rts             ;basic
;
;------- basicanf ---------------
;
cbb7 ad55c8 :6010 -basic      lda prg         ;basicanf
cbba ac56c8 :6020 -           ldy prg+1       ;auf prg-
cbbd 852b   :6030 -           sta 43          ;start
cbbf 842c   :6040 -           sty 44
cbc1 60     :6050 -           rts             ;basic
;
;--- anfang bs-/farbspeicher ----
;
cbc2 20a3cb :6090 -screen     jsr parameter+3;anfang
cbc5 8d51c8 :6100 -           sta bsconst     ;bildsch.
cbc8 8c52c8 :6110 -           sty bsconst+1
cbcb 60     :6120 -           rts
;
;------- speicherbelegung --------
```

Listing 4/6.3.9.3-1 (Teil 12)

```
;
cbcc a920    :6160 -mem       lda #32        ;leerz.
cbce 20d2ff  :6170 -          jsr bsout      ;ausgeben
cbd1 ae48c8  :6180 -          ldx anf1       ;speicher-
cbd4 ad49c8  :6190 -          lda anf2       ;anfang
cbd7 20cdbd  :6200 -          jsr linprt     ;ausgeben
;
cbda a92d    :6220 -          lda #"-"       ;trennz.
cbdc 20d2ff  :6230 -          jsr bsout
cbdf ae4ec8  :6240 -          ldx spend      ;zaehler
cbe2 ad4fc8  :6250 -          lda spend+1    ;ausgeben
cbe5 20cdbd  :6260 -          jsr linprt
cbe8 a90d    :6270 -          lda #13        ;return
cbea 20d2ff  :6280 -          jsr bsout
cbed 60      :6290 -          rts            ;basic
;
;---- farbe der leerzeichen -----
;
cbee 209eb7  :6330 -leer      jsr getbyte    ;farbe
cbf1 8e42c8  :6340 -          stx spcol      ;setzen
cbf4 60      :6350 -          rts            ;basic
;
;------- anzahl bs-ausgeben -----
;
cbf5 a920    :6390 -anz       lda #32        ;leerz.
cbf7 20d2ff  :6400 -          jsr bsout      ;ausgeben
cbfa ae50c8  :6410 -          ldx anzahl     ;speicher-
cbfd a900    :6420 -          lda #0         ;anfang
cbff 20cdbd  :6430 -          jsr linprt     ;ausgeben
cc02 a90d    :6440 -          lda #13        ;return
cc04 20d2ff  :6450 -          jsr bsout
cc07 60      :6460 -          rts            ;basic
;
;---------------------------------
;--erweiterung der interpreters.-
;---------------------------------
;
;------ befehlsauswertung -------
;
cc08 207300  :6540 -inter     jsr chrget     ;zeichen
cc0b c960    :6550 -          cmp #$60       ;holen
cc0d b019    :6560 -          bcs interout   ;kein
cc0f c941    :6570 -          cmp #$41       ;buch-
cc11 9015    :6580 -          bcc interout   ;stabe ?
cc13 8d47c8  :6590 -          sta akku       ;zeichen
cc16 a200    :6600 -          ldx #00        ;sichern
cc18 8e6bc8  :6610 -          stx befnr      ;zaehler=0
cc1b a000    :6620 -inter1    ldy #00
cc1d ee6bc8  :6630 -          inc befnr      ;befehls-
cc20 bd8ccc  :6640 -          lda befehle,x  ;tabelle
cc23 d009    :6650 -          bne inter2     ;auslesen
;
cc25 ad47c8  :6670 -          lda akku       ;zurueck
cc28 207900  :6680 -interout  jsr chrgot     ;zum
```

Listing 4/6.3.9.3-1 (Teil 13)

```
cc2b 4ce7a7 :6690 -          jmp intend      ;BASIC
;
;------ adresse suchen ----------
;
cc2e d17a   :6730 -inter2    cmp (txtptr),y ;basicbef
cc30 d035   :6740 -          bne rest        ;=tabelle?
cc32 e8     :6750 -          inx             ;zaehler+1
cc33 c8     :6760 -          iny
cc34 bd8ccc :6770 -          lda befehle,x   ;letztes
cc37 c980   :6780 -          cmp #128        ;zeichen ?
cc39 90f3   :6790 -          bcc inter2
cc3b 297f   :6800 -          and #127        ;bit 7
cc3d d17a   :6810 -          cmp (txtptr),y ;loeschen
cc3f f004   :6820 -          beq inter3      ;auch ok ?
cc41 e8     :6830 -          inx             ;nein,
cc42 4c1bcc :6840 -          jmp inter1      ;weiter
;
cc45 18     :6860 -inter3    clc             ;txtptr
cc46 c8     :6870 -          iny             ;auf
cc47 98     :6880 -          tya             ;aktuellen
cc48 657a   :6890 -          adc txtptr      ;stand
cc4a 857a   :6900 -          sta txtptr      ;bringen
cc4c 9002   :6910 -          bcc lab1
cc4e e67b   :6920 -          inc txtptr+1
;
cc50 ad6bc8 :6940 -lab1      lda befnr       ;befnr *2
cc53 0a     :6950 -          asl
cc54 aa     :6960 -          tax
cc55 bd75cc :6970 -          lda befadr,x    ;sprung-
cc58 bc76cc :6980 -          ldy befadr1,x   ;tabelle
cc5b 8d62cc :6990 -          sta jump+1      ;auslesen
cc5e 8c63cc :7000 -          sty jump+2      ;und neue
cc61 20ffff :7010 -jump      jsr $ffff       ;routine
cc64 4c28cc :7020 -          jmp interout
;
;---- befehlstext ueberlesen ----
;
cc67 e8     :7060 -rest      inx             ;zaehler+1
cc68 bd8ccc :7070 -          lda befehle,x   ;letztes
cc6b 2980   :7080 -          and #128        ;zeichen
cc6d c900   :7090 -          cmp #0          ;der tab.?
cc6f f0f6   :7100 -          beq rest        ;nein.
cc71 e8     :7110 -          inx             ;ja,
cc72 4c1bcc :7120 -          jmp inter1      ;weiter
;
;--------------------------------
;-- sprungtabelle neue befehle --
;--------------------------------
;
             7180 -befadr    .by $e7
             7190 -befadr1   .by $a7
             7200 -          .wo deconvert
             7210 -          .wo leer
             7220 -          .wo load
```

Listing 4/6.3.9.3-1 (Teil 14)

```
                7230 -           .wo mem
                7240 -           .wo anz
                7250 -           .wo screen
                7260 -           .wo norm
                7270 -           .wo basic
                7280 -           .wo prganf
                7290 -           .wo interoff
                7300 -           .by 0
;
;--------------------------------
;--        befehlstabelle      --
;--------------------------------
;
                7360 -befehle    .tx "bS"
                7370 -           .tx "spacE"
                7380 -           .tx "disK"
                7390 -           .tx "meM"
                7400 -           .tx "nuM"
                7410 -           .tx "screeN"
                7420 -           .tx "nrM"
                7430 -           .tx "prG"
                7440 -           .tx "basiC"
                7450 -           .tx "ofF"
                7460 -           .by 0
;
;-------- startflag -------------
;
ccb2 a502    :7500 -extend     lda start       ;erster
ccb4 f003    :7510 -           beq extend2     ;start ?
ccb6 4c6dc8  :7520 -           jmp init        ;nein
;
ccb9 a901    :7540 -extend2    lda #1          ;texte im
ccbb 8502    :7550 -           sta start       ;programm
ccbd ad55c8  :7560 -           lda prg         ;einge-
ccc0 cd4ec8  :7570 -           cmp spend       ;bunden ?
ccc3 d00b    :7580 -           bne extend3
ccc5 ad56c8  :7590 -           lda prg+1
ccc8 cd4fc8  :7600 -           cmp spend+1
cccb d003    :7610 -           bne extend3
cccd 4c6dc8  :7620 -           jmp init        ;ja
;
ccd0 a900    :7640 -extend3    lda #0          ;bs-anzahl
ccd2 8d50c8  :7650 -           sta anzahl      ;loeschen
ccd5 4c6dc8  :7660 -           jmp init        ;weiter
;
;------- basicanfang ------------
;
                7700 -basicanf   .by 00          ;Null-
                7710 -           .by 00          ;bytes
                7720 -           .by 00          ;fuer
                7730 -           .by 00          ;basicanf.
```

Listing 4/6.3.9.3-1 (Teil 15)

```
hypra-ass  assemblerlisting:

              0    -.li 1,4,7
;********************************
;**                            **
;**  TEXT - CONVERT - SYSTEM   **
;**                            **
;**     Quelltext Converter    **
;**                            **
;********************************
;**                            **
;********************************
;
;
;------ variablen extern --------
;
              200   -.eq bs=$f7
              210   -.eq sp=$f9
              220   -.eq cl=$fb
              230   -.eq tb=$fd
;
;--------- konstanten ----------
;
              270   -.eq revon     = 128
              280   -.eq revoff    = 129
              290   -.eq space     =  32
              300   -.eq linend    = 146
              310   -.eq ctrl      = 653
;
;---------- routinen -----------
;
              350   -.eq keyvec    = $ 28f
              360   -.eq oldkey    = $eb48
              370   -.eq milli     = $eeb3
              380   -.eq get       = $ffe4
              390   -.eq open      = $ffc0
              400   -.eq setfls    = $ffba
              410   -.eq setnam    = $ffbd
              420   -.eq close     = $ffc3
              430   -.eq ckout     = $ffc9
              440   -.eq clrch     = $ffcc
              450   -.eq bsout     = $ffd2
              460   -.eq chkcom    = $aefd
              470   -.eq getbyte   = $b79e
              480   -.eq chrget    = $   73
              490   -.eq chrgot    = $   79
              500   -.eq error     = $a437
              510   -.eq illquan   = $b248
              520   -.eq linprt    = $bdcd
              530   -.eq ready     = $e37b
              540   -.eq frmnum    = $ad8a
```

Listing 4/6.3.9.3-2 (Teil 1)

```
              550  -.eq integer   = $b7f7
              560  -.eq intvec    = $0308
              570  -.eq intend    = $a7e7
              580  -.eq interalt  = $a7e4
              590  -.eq txtptr    = $7a
              600  -.eq errout    = $a445
;
;
              630  -.ba 39300
;
;----------------------------------
;--    befehlsvector umlegen    --
;----------------------------------
;
9984 4ca699 :690  -stprg      jmp spgrenze
9987 a94b   :700  -stprg1     lda #<(inter)  ;befehls-
9989 a09f   :710  -           ldy #>(inter)  ;vector
998b 8d0803 :720  -           sta intvec     ;auf neue
998e 8c0903 :730  -           sty intvec+1   ;routine
;
9991 adf699 :750  -           lda stflag     ;zum 1.mal
9994 f061   :760  -           beq init       ;gestartet
9996 4c449a :770  -           jmp vecon      ;basic
;
9999 a9e4   :790  -interoff   lda #<(interalt)
999b a0a7   :800  -           ldy #>(interalt)
999d 8d0803 :810  -           sta intvec
99a0 8c0903 :820  -           sty intvec+1
99a3 4c4f9a :830  -           jmp vecoff
;
;------ speicher begrenzen ------
;
99a6 a984   :870  -spgrenze   lda #<(stprg)
99a8 a099   :880  -           ldy #>(stprg)
99aa 8537   :890  -           sta 55
99ac 8438   :900  -           sty 56
;
99ae a52d   :920  -           lda 45
99b0 a42e   :930  -           ldy 46
99b2 852f   :940  -           sta 47
99b4 8531   :950  -           sta 49
99b6 8430   :960  -           sty 48
99b8 8432   :970  -           sty 50
99ba 4c8799 :980  -           jmp stprg1
;
;----------------------------------
;--    variablen prg-intern     --
;----------------------------------
;
              1040 -code       .by 00
              1050 -code2      .by 00
```

Listing 4/6.3.9.3-2 (Teil 2)

```
                  1060 -revflag    .by 00
                  1070 -line       .by 00
                  1080 -line2      .by 00
                  1090 -col        .by 00
                  1100 -yreg       .by 00
                  1110 -count      .by 00
                  1120 -pruef      .by 00
                  1130 -flag1      .by 00
                  1140 -flag2      .by 00
                  1150 -flag3      .by 00
                  1160 -zws        .wo 00
                  1170 -zws2       .wo 00
                  1180 -akku       .by 00
                  1190 -namec      .wo 00
                  1200 -bscol      .by 00
                  1210 -spconst    .wo 00
                  1220 -bsconst    .wo $0400
                  1230 -clconst    .wo $d800
                  1240 -konf       .by 00
                  1250 -tbalt      .wo 00
                  1260 -spalt      .wo 00
                  1270 -tbconst    .wo 00
                  1280 -spnew      .wo 00
                  1290 -tbwert     .wo 00
                  1300 -spanf      .wo $a000
                  1310 -befnr      .by 00
                  1320 -errmsg     .tx "no screen defineD"
                  1330 -stflag     .by 00
;
;--------------------------------
;--     initialisierung       --
;--------------------------------
;
99f7 a901    :1390 -init      lda #1         ;1. start
99f9 8df699  :1400 -          sta stflag
99fc a900    :1410 -          lda #0         ;anzahl
99fe 8dce99  :1420 -          sta namec      ;screens=0
9a01 8de099  :1430 -          sta tbwert
9a04 8de199  :1440 -          sta tbwert+1
;
9a07 ade299  :1460 -          lda spanf      ;sp
9a0a ace399  :1470 -          ldy spanf+1    ;anfangs-
9a0d 85f9    :1480 -          sta sp         ;wert in
9a0f 8dd899  :1490 -          sta tbalt      ;zeigern
9a12 8ddc99  :1500 -          sta tbconst
9a15 8dd199  :1510 -          sta spconst
9a18 84fa    :1520 -          sty sp+1       ;speichern
9a1a 8cd999  :1530 -          sty tbalt+1
9a1d 8cdd99  :1540 -          sty tbconst+1
9a20 8cd299  :1550 -          sty spconst+1
;
```

Listing 4/6.3.9.3-2 (Teil 3)

```
9a23 20419d :1570 -             jsr ram
9a26 20419d :1580 -             jsr ram
9a29 a0ff   :1590 -             ldy #255          ;sprung-
9a2b 20419d :1600 -init1        jsr ram           ;tabelle
9a2e 20419d :1610 -             jsr ram           ;ein-
9a31 88     :1620 -             dey               ;richten
9a32 d0f7   :1630 -             bne init1         ;&loeschen
;
9a34 a5f9   :1650 -             lda sp            ;anfang
9a36 a4fa   :1660 -             ldy sp+1          ;text-
9a38 8dda99 :1670 -             sta spalt         ;speicher
9a3b 8dd199 :1680 -             sta spconst       ;merken
9a3e 8cdb99 :1690 -             sty spalt+1
9a41 8cd299 :1700 -             sty spconst+1
;
;---- tastaturvector umbiegen ---
;
9a44 a95a   :1740 -vecon        lda #<(test)      ;tastatur-
9a46 a09a   :1750 -             ldy #>(test)      ;vector
9a48 8d8f02 :1760 -             sta keyvec        ;auf neue
9a4b 8c9002 :1770 -             sty keyvec+1      ;routine
9a4e 60     :1780 -             rts
;
9a4f a948   :1800 -vecoff       lda #<(oldkey)    ;vector
9a51 a0eb   :1810 -             ldy #>(oldkey)    ;zurueck
9a53 8d8f02 :1820 -             sta keyvec
9a56 8c9002 :1830 -             sty keyvec+1
9a59 60     :1840 -             rts
;
;------- taste gedrueckt --------
;
9a5a ad8d02 :1880 -test         lda ctrl          ;ctrl
9a5d 2904   :1890 -             and #4            ;gedrueckt
9a5f f006   :1900 -             beq no
9a61 a5c5   :1910 -             lda $c5
9a63 c939   :1920 -             cmp #57           ;← gedr.?
9a65 f003   :1930 -             beq flimmer
;
9a67 4c48eb :1950 -no           jmp oldkey
;
;--------------------------------
;--          bs-flimmern       --
;--------------------------------
;
9a6a ad20d0 :2010 -flimmer      lda $d020         ;rahmenf.
9a6d 8dd099 :2020 -             sta bscol         ;sichern
9a70 a5cf   :2030 -cur1         lda 207           ;cursor
9a72 eec599 :2040 -             inc pruef         ;aus
9a75 f0f9   :2050 -             beq cur1
;
9a77 a901   :2070 -             lda #1
```

Listing 4/6.3.9.3-2 (Teil 4)

```
9a79 85cc   :2080 -           sta 204
9a7b a900   :2090 -           lda #0          ;taste
9a7d 85c6   :2100 -           sta 198         ;loeschen
9a7f 204f9a :2110 -           jsr vecoff
9a82 58     :2120 -           cli
9a83 a064   :2130 -           ldy #100        ;kurz
9a85 20be9a :2140 -           jsr wait        ;warten
;
9a88 a906   :2160 -flimmer1   lda #6          ; blau
9a8a 8d20d0 :2170 -           sta $d020
9a8d a004   :2180 -           ldy #4          ;warten
9a8f 20be9a :2190 -           jsr wait
;
9a92 a90e   :2210 -           lda #14         ;hellblau
9a94 8d20d0 :2220 -           sta $d020
9a97 a091   :2230 -           ldy #145        ;warten
9a99 88     :2240 -flimmer2   dey
9a9a d0fd   :2250 -           bne flimmer2
9a9c a005   :2260 -           ldy #5
9a9e 20be9a :2270 -           jsr wait
;
9aa1 20e4ff :2290 -           jsr get         ;taste ge-
9aa4 f0e2   :2300 -           beq flimmer1    ;drueckt ?
;
9aa6 c920   :2320 -           cmp #32         ;space ?
9aa8 f007   :2330 -           beq flimmer3    ;ja
9aaa c90d   :2340 -           cmp #13         ;return ?
9aac f017   :2350 -           beq convert     ;ja
9aae 4c889a :2360 -           jmp flimmer1
;
9ab1 a900   :2380 -flimmer3   lda #0          ;taste
9ab3 85c6   :2390 -           sta 198
9ab5 add099 :2400 -           lda bscol       ;rahmenf.
9ab8 8d20d0 :2410 -           sta $d020
9abb 4c449a :2420 -           jmp vecon       ;basic
;
9abe 20b3ee :2440 -wait       jsr milli       ;y mal
9ac1 88     :2450 -           dey             ;1 milli
9ac2 d0fa   :2460 -           bne wait        ;sekunde
9ac4 60     :2470 -           rts
;
;----------------------------------
;--          prg- init          --
;----------------------------------
;
9ac5 a900   :2530 -convert    lda #0          ;tastatur-
9ac7 85c6   :2540 -           sta 198         ;puffer=0
;
9ac9 add199 :2560 -           lda spconst     ;speicher
9acc acd299 :2570 -           ldy spconst+1   ;zaehler
9acf 85f9   :2580 -           sta sp          ;anfang
```

Listing 4/6.3.9.3-2 (Teil 5)

```
9ad1 84fa   :2590 -           sty sp+1
;
9ad3 addc99 :2610 -           lda tbconst    ;tabellen-
9ad6 acdd99 :2620 -           ldy tbconst+1  ;zaehler
9ad9 85fd   :2630 -           sta tb
9adb 84fe   :2640 -           sty tb+1
;
9add add399 :2660 -           lda bsconst    ;anfang
9ae0 acd499 :2670 -           ldy bsconst+1  ;bild-
9ae3 85f7   :2680 -           sta bs         ;schirm-
9ae5 84f8   :2690 -           sty bs+1       ;speicher
;
9ae7 add599 :2710 -           lda clconst    ;anfang
9aea acd699 :2720 -           ldy clconst+1  ;farb-
9aed 85fb   :2730 -           sta cl         ;speicher
9aef 84fc   :2740 -           sty cl+1
;
9af1 eec899 :2760 -           inc flag3      ;text-
9af4 ade099 :2770 -           lda tbwert     ;anfang
9af7 20419d :2780 -           jsr ram        ;in die
9afa ade199 :2790 -           lda tbwert+1   ;tabelle
9afd 20419d :2800 -           jsr ram        ;eintragen
9b00 cec899 :2810 -           dec flag3
9b03 18     :2820 -           clc
;
9b04 a900   :2840 -           lda #0         ;diverse
9b06 8dbf99 :2850 -           sta revflag    ;flags
9b09 8dc299 :2860 -           sta col        ;auf 0
9b0c 8dc099 :2870 -           sta line
9b0f 8dc399 :2880 -           sta yreg
;
9b12 a906   :2900 -           lda #6         ;rahmenf.
9b14 8d20d0 :2910 -           sta $d020      ;blau
;
9b17 adce99 :2930 -           lda namec      ;256.
9b1a c9ff   :2940 -           cmp #255       ;screen ?
9b1c d005   :2950 -           bne convert1
;
9b1e a210   :2970 -           ldx #16        ;out of
9b20 4c37a4 :2980 -           jmp error      ;memory
;
9b23 eece99 :3000 -convert1   inc namec      ;textnr.+1
;
;---------- bs-farbe -------------
;
9b26 adc799 :3040 -           lda flag2      ;keine
9b29 d011   :3050 -           bne loop2      ;farbe ?
;
9b2b a9ff   :3070 -           lda #255       ;signal-
9b2d 20419d :3080 -           jsr ram        ;byte
9b30 add099 :3090 -           lda bscol      ;rahmen-
```

Listing 4/6.3.9.3-2 (Teil 6)

```
9b33 20419d :3100 -              jsr ram         ;farbe
9b36 ad21d0 :3110 -              lda $d021       ;bilds.
9b39 20419d :3120 -              jsr ram         ;farbe
;
;--------------------------------
;--        hauptschleife       --
;--------------------------------
;
9b3c a000   :3180 -loop2         ldy #0
9b3e b1f7   :3190 -loop1         lda (bs),y      ;bs-code
9b40 8dbd99 :3200 -              sta code        ;sichern
;
9b43 c980   :3220 -              cmp #128        ;reverses
9b45 9003   :3230 -              bcc weiter2     ;zeichen ?
9b47 4cde9b :3240 -              jmp rflag
;
9b4a adbf99 :3260 -weiter2       lda revflag     ;revflag
9b4d f003   :3270 -              beq weiter      ;trotzdem
9b4f 4c0c9c :3280 -              jmp rflagb      ;gesetzt
;
9b52 adbd99 :3300 -weiter        lda code        ;leer-
9b55 29bf   :3310 -              and #191        ;zeichen ?
9b57 c920   :3320 -              cmp #32
9b59 d006   :3330 -              bne save
9b5b 8dbd99 :3340 -              sta code
9b5e 4c6d9b :3350 -              jmp save2
;
;
9b61 b1fb   :3380 -save          lda (cl),y      ;farbcode
9b63 290f   :3390 -              and #15         ;holen
9b65 cdc299 :3400 -              cmp col         ;neue
9b68 f003   :3410 -              beq save2       ;farbe ?
9b6a 4c219c :3420 -              jmp colneu
;
9b6d 4c399c :3440 -save2         jmp double
9b70 8cc399 :3450 -save3         sty yreg        ;y sichern
9b73 a000   :3460 -              ldy #0
9b75 adbd99 :3470 -              lda code        ;code
9b78 20419d :3480 -              jsr ram         ;speichern
9b7b acc399 :3490 -              ldy yreg
;
;
9b7e c8     :3520 -zaehler       iny             ;zaehler+1
9b7f c028   :3530 -zaehler2      cpy #40         ;zeilen-
9b81 d0bb   :3540 -              bne loop1       ;ende ?
;
9b83 a5f7   :3560 -newline       lda bs          ;naechste
9b85 18     :3570 -              clc             ;zeile
9b86 6928   :3580 -              adc #40         ;bild-
9b88 85f7   :3590 -              sta bs          ;schirmsp.
9b8a a5f8   :3600 -              lda bs+1
```

Listing 4/6.3.9.3-2 (Teil 7)

```
9b8c 6900    :3610 -            adc #0
9b8e 85f8    :3620 -            sta bs+1
9b90 18      :3630 -            clc
;
9b91 a5fb    :3650 -            lda cl          ;naechste
9b93 18      :3660 -            clc             ;zeile
9b94 6928    :3670 -            adc #40         ;farbsp.
9b96 85fb    :3680 -            sta cl
9b98 a5fc    :3690 -            lda cl+1
9b9a 6900    :3700 -            adc #0
9b9c 85fc    :3710 -            sta cl+1
9b9e 18      :3720 -            clc
9b9f adc699  :3730 -            lda flag1
9ba2 f001    :3740 -            beq newline1
9ba4 60      :3750 -            rts
;
9ba5 eec099  :3770 -newline1    inc line        ;letzte
9ba8 adc099  :3780 -newline2    lda line        ;zeile ?
9bab c919    :3790 -            cmp #25
9bad d08d    :3800 -            bne loop2
;
9baf add099  :3820 -ende        lda bscol       ;rahmenf.
9bb2 8d20d0  :3830 -            sta $d020       ;zurueck
9bb5 a5f9    :3840 -            lda sp          ;zaehler
9bb7 a4fa    :3850 -            ldy sp+1        ;sichern
9bb9 8dd199  :3860 -            sta spconst
9bbc 8cd299  :3870 -            sty spconst+1
;
9bbf a5f9    :3890 -            lda sp          ;speicher-
9bc1 38      :3900 -            sec             ;zaehler
9bc2 edda99  :3910 -            sbc spalt       ;vom
9bc5 8de099  :3920 -            sta tbwert      ;anfang
9bc8 a5fa    :3930 -            lda sp+1        ;abziehen
9bca eddb99  :3940 -            sbc spalt+1
9bcd 8de199  :3950 -            sta tbwert+1
9bd0 18      :3960 -            clc
;
9bd1 a5fd    :3980 -            lda tb          ;tabellen-
9bd3 a4fe    :3990 -            ldy tb+1        ;zeiger
9bd5 8ddc99  :4000 -            sta tbconst     ;sichern
9bd8 8cdd99  :4010 -            sty tbconst+1
9bdb 4c449a  :4020 -            jmp vecon
;
;------ reverse zeichen ---------
;
9bde adbd99  :4060 -rflag       lda code        ;reverses
9be1 38      :4070 -            sec             ;zeichen
9be2 e980    :4080 -            sbc #128        ;normali-
9be4 8dbd99  :4090 -            sta code        ;sieren
9be7 18      :4100 -            clc
;
```

Listing 4/6.3.9.3-2 (Teil 8)

```
9be8 c960    :4120 -            cmp #96        ;2.space ?
9bea d005    :4130 -            bne rflag2
9bec a920    :4140 -            lda #32        ;dann code
9bee 8dbd99  :4150 -            sta code       ;= 32
;
;
9bf1 adbf99  :4180 -rflag2      lda revflag    ;revflag
9bf4 f003    :4190 -            beq rflag1     ;=1 ?
9bf6 4c619b  :4200 -            jmp save
9bf9 eebf99  :4210 -rflag1      inc revflag    ;revflag=1
9bfc 8cc399  :4220 -            sty yreg       ;y sichern
9bff a000    :4230 -            ldy #0
9c01 a980    :4240 -            lda #revon     ;zeichen
9c03 20419d  :4250 -            jsr ram        ;in den
9c06 acc399  :4260 -            ldy yreg       ;speicher
9c09 4c619b  :4270 -            jmp save       ;zurueck
;
;--------- normale zeichen ------
;
9c0c a900    :4310 -rflagb      lda #0         ;revflag=0
9c0e 8dbf99  :4320 -            sta revflag
9c11 8cc399  :4330 -            sty yreg       ;y sichern
9c14 a000    :4340 -            ldy #0
9c16 a981    :4350 -            lda #revoff    ;zeichen
9c18 20419d  :4360 -            jsr ram        ;in den
9c1b acc399  :4370 -            ldy yreg       ;speicher
9c1e 4c529b  :4380 -            jmp weiter     ;zurueck
;
;-------- farbe aendern ---------
;
9c21 b1fb    :4420 -colneu      lda (cl),y
9c23 290f    :4430 -            and #15
9c25 8dc299  :4440 -            sta col        ;neue
9c28 18      :4450 -            clc
9c29 6982    :4460 -            adc #130       ;farbe
9c2b 8cc399  :4470 -            sty yreg
9c2e a000    :4480 -            ldy #0         ;farbcode
9c30 20419d  :4490 -            jsr ram        ;in den
9c33 acc399  :4500 -            ldy yreg       ;speicher
9c36 4c6d9b  :4510 -            jmp save2      ;zurueck
;
;------ gleiche zeichen ---------
;
9c39 8cc399  :4550 -double      sty yreg       ;y sichern
9c3c a901    :4560 -            lda #1         ;zaehler=1
9c3e 8dc499  :4570 -            sta count
9c41 adbd99  :4580 -            lda code
9c44 8dbe99  :4590 -            sta code2
;
9c47 adbf99  :4610 -            lda revflag    ;reverses
9c4a f009    :4620 -            beq double5    ;zeichen ?
```

Listing 4/6.3.9.3-2 (Teil 9)

```
9c4c adbe99 :4630 -              lda code2       ;normaler
9c4f 6980   :4640 -              adc #128        ;zeichen-
9c51 8dbe99 :4650 -              sta code2       ;code
9c54 18     :4660 -              clc
;
9c55 c027   :4680 -double5       cpy #39         ;letztes
9c57 f022   :4690 -              beq double4     ;zeichen ?
;
9c59 c8     :4710 -double2       iny             ;naechstes
9c5a b1f7   :4720 -              lda (bs),y      ;zeichen
9c5c cdbe99 :4730 -              cmp code2       ;gleich ?
9c5f d023   :4740 -              bne double3
9c61 29bf   :4750 -              and #191
9c63 c920   :4760 -              cmp #32
9c65 f009   :4770 -              beq double6
9c67 b1fb   :4780 -              lda (cl),y      ;farbe
9c69 290f   :4790 -              and #15
9c6b cdc299 :4800 -              cmp col         ;gleich
9c6e d014   :4810 -              bne double3
9c70 18     :4820 -double6       clc
;
9c71 eec499 :4840 -              inc count
9c74 c027   :4850 -              cpy #39         ;zeilen-
9c76 d0e1   :4860 -              bne double2     ;ende ?
9c78 cec499 :4870 -              dec count
;
9c7b adbe99 :4890 -double4       lda code2       ;nur leer-
9c7e 29bf   :4900 -              and #191
9c80 c920   :4910 -              cmp #32         ;zeichen ?
9c82 f029   :4920 -              beq zendex
;
9c84 adc499 :4940 -double3       lda count       ;4 gleiche
9c87 c904   :4950 -              cmp #4          ;zeichen
9c89 901c   :4960 -              bcc doublend
9c8b 8cc399 :4970 -              sty yreg
9c8e a000   :4980 -              ldy #0
9c90 a993   :4990 -              lda #147        ;signalc.
9c92 20419d :5000 -              jsr ram         ;speichern
;
9c95 adc499 :5020 -              lda count       ;anzahl
9c98 20419d :5030 -              jsr ram         ;speichern
;
9c9b adbe99 :5050 -              lda code2       ;zeichen
9c9e 20419d :5060 -              jsr ram         ;speichern
9ca1 acc399 :5070 -              ldy yreg
9ca4 4c7f9b :5080 -              jmp zaehler2
;
9ca7 acc399 :5100 -doublend      ldy yreg        ;y zurueck
9caa 4c709b :5110 -              jmp save3       ;weiter
;
;----- ende der zeile -----------
```

Listing 4/6.3.9.3-2 (Teil 10)

```
;
9cad a000   :5150 -zendex     ldy #0         ;code fuer
9caf 4cba9c :5160 -           jmp bsend      ;in den
9cb2 a992   :5170 -zendex1    lda #linend    ;return
9cb4 20419d :5180 -           jsr ram
9cb7 4c839b :5190 -           jmp newline    ;neue z.
;
;----- mehrere leerzeilen -------
;
9cba a5f7   :5230 -bsend      lda bs         ;text-
9cbc 8dc999 :5240 -           sta zws        ;zeiger
9cbf a5f8   :5250 -           lda bs+1       ;zwischen-
9cc1 8dca99 :5260 -           sta zws+1      ;speichern
;
9cc4 a5fb   :5280 -           lda cl         ;farb-
9cc6 8dcb99 :5290 -           sta zws2       ;speicher
9cc9 a5fc   :5300 -           lda cl+1       ;zwischen-
9ccb 8dcc99 :5310 -           sta zws2+1     ;speichern
;
9cce a000   :5330 -           ldy #0         ;zaehler=0
9cd0 8cc499 :5340 -           sty count
9cd3 c8     :5350 -           iny
9cd4 8cc699 :5360 -           sty flag1
9cd7 adc099 :5370 -           lda line       ;zeilennr.
9cda 8dc199 :5380 -           sta line2      ;sichern
;
9cdd eec099 :5400 -           inc line       ;schon
9ce0 adc099 :5410 -           lda line       ;letzte
9ce3 c919   :5420 -           cmp #25        ;zeile ?
9ce5 f00d   :5430 -           beq bsend4     ;ja
;
9ce7 a000   :5450 -bsend1     ldy #0         ;neue
9ce9 20839b :5460 -           jsr newline    ;zeile ?
9cec b1f7   :5470 -bsend2     lda (bs),y
9cee 29bf   :5480 -           and #191       ;leer-
9cf0 c920   :5490 -           cmp #32        ;zeichen ?
9cf2 f027   :5500 -           beq bsend3     ;ja
;
9cf4 adc499 :5520 -bsend4     lda count      ;>0 leer-
9cf7 d037   :5530 -           bne bsend5     ;zeilen ?
;
9cf9 a000   :5550 -           ldy #0         ;zeiger
9cfb 8cc699 :5560 -           sty flag1      ;zurueck
9cfe adc999 :5570 -           lda zws
9d01 85f7   :5580 -           sta bs
9d03 adca99 :5590 -           lda zws+1
9d06 85f8   :5600 -           sta bs+1
;
9d08 adcb99 :5620 -           lda zws2
9d0b 85fb   :5630 -           sta cl
9d0d adcc99 :5640 -           lda zws2+1
```

Listing 4/6.3.9.3-2 (Teil 11)

```
9d10 85fc    :5650 -          sta cl+1
;
9d12 adc199 :5670 -          lda line2
9d15 8dc099 :5680 -          sta line
9d18 4cb29c :5690 -          jmp zendex1      ;alte r.
;
9d1b c8      :5710 -bsend3    iny              ;naechstes
9d1c c028    :5720 -          cpy #40          ;zeichen
9d1e d0cc    :5730 -          bne bsend2
;
9d20 eec499 :5750 -          inc count        ;naechste
9d23 eec099 :5760 -          inc line
9d26 adc099 :5770 -          lda line         ;zeile
9d29 c919    :5780 -          cmp #25          ;letzte ?
9d2b d0ba    :5790 -          bne bsend1
;           dec count
9d2d 4cf49c :5810 -          jmp bsend4       ;weiter
;
9d30 a000    :5830 -bsend5    ldy #0           ;zeiger
9d32 8cc699 :5840 -          sty flag1        ;zurueck
9d35 a994    :5850 -          lda #148         ;signal-
9d37 18      :5860 -          clc              ;code
9d38 6dc499 :5870 -          adc count        ;speichern
9d3b 20419d :5880 -          jsr ram
9d3e 4ca89b :5890 -          jmp newline2
;
;------ zeichen ins ram ---------
;
9d41 8dcd99 :5930 -ram       sta akku         ;register
9d44 8a      :5940 -          txa              ;retten
9d45 48      :5950 -          pha
9d46 98      :5960 -          tya
9d47 48      :5970 -          pha
;
9d48 a501    :5990 -          lda 1            ;speicher
9d4a 8dd799 :6000 -          sta konf         ;auf ram
9d4d 29f8    :6010 -          and #248         ;schalten
9d4f 78      :6020 -          sei
9d50 8501    :6030 -          sta 1
9d52 adc899 :6040 -          lda flag3        ;tabellen-
9d55 d02c    :6050 -          bne ram3         ;wert ?
9d57 a000    :6060 -          ldy #0
9d59 adcd99 :6070 -          lda akku         ;zeichen
9d5c 91f9    :6080 -          sta (sp),y       ;speichern
9d5e acd799 :6090 -          ldy konf
9d61 8401    :6100 -          sty 1
9d63 58      :6110 -          cli
9d64 18      :6120 -          clc
;
9d65 e6f9    :6140 -          inc sp           ;zaehler
9d67 d002    :6150 -          bne ram1         ;erhoehen
```

Listing 4/6.3.9.3-2 (Teil 12)

```
9d69 e6fa   :6160 -           inc sp+1
;
9d6b a5f9   :6180 -ram1       lda sp         ;speicher-
9d6d c9ff   :6190 -           cmp #$ff       ;grenze
9d6f a5fa   :6200 -           lda sp+1       ;erreicht?
9d71 e9ff   :6210 -           sbc #$ff
9d73 9005   :6220 -           bcc ram2
;
9d75 a210   :6240 -           ldx #16        ;out of
9d77 4c37a4 :6250 -           jmp error      ;mem.error
;
9d7a 68     :6270 -ram2       pla            ;register
9d7b a8     :6280 -           tay            ;zurueck
9d7c 68     :6290 -           pla
9d7d aa     :6300 -           tax
9d7e adcd99 :6310 -           lda akku
9d81 18     :6320 -           clc
9d82 60     :6330 -           rts            ;zurueck
;
9d83 a000   :6350 -ram3       ldy #0         ;sprung-
9d85 adcd99 :6360 -           lda akku       ;adresse
9d88 91fd   :6370 -           sta (tb),y     ;in der
9d8a acd799 :6380 -           ldy konf       ;tabelle
9d8d 8401   :6390 -           sty 1          ;speichern
9d8f 58     :6400 -           cli
9d90 18     :6410 -           clc
;
9d91 e6fd   :6430 -           inc tb         ;tabellen-
9d93 d0e5   :6440 -           bne ram2       ;zaehler
9d95 e6fe   :6450 -           inc tb+1       ;erhoehen
9d97 4c7a9d :6460 -           jmp ram2       ;weiter
;
;--------------------------------
;-parameter abfragen/einstellen -
;--------------------------------
;
;------ bs-anzahl ausgeben ------
;
9d9a a920   :6540 -anz        lda #32        ;leerz.
9d9c 20d2ff :6550 -           jsr bsout      ;ausgeben
9d9f aece99 :6560 -           ldx namec      ;zaehler
9da2 a900   :6570 -           lda #0         ;ausgeben
9da4 20cdbd :6580 -           jsr linprt     ;return
9da7 a90d   :6590 -           lda #13        ;ausgeben
9da9 4cd2ff :6600 -           jmp bsout      ;basic
;
;----- sp-zaehler ausgeben ------
;
9dac a920   :6640 -mem        lda #32        ;leerz.
9dae 20d2ff :6650 -           jsr bsout      ;ausgeben
9db1 aee299 :6660 -           ldx spanf      ;speicher-
```

Listing 4/6.3.9.3-2 (Teil 13)

```
9db4 ade399 :6670 -              lda spanf+1     ;anfang
9db7 20cdbd :6680 -              jsr linprt      ;ausgeben
;
9dba a92d   :6700 -              lda #"-"        ;trennz.
9dbc 20d2ff :6710 -              jsr bsout       ;ausgeben
9dbf aed199 :6720 -              ldx spconst     ;zaehler
9dc2 add299 :6730 -              lda spconst+1   ;ausgeben
9dc5 20cdbd :6740 -              jsr linprt
9dc8 a90d   :6750 -              lda #13         ;return
9dca 20d2ff :6760 -              jsr bsout       ;ausgeben
9dcd 60     :6770 -              rts             ;basic
;
;-------- speicheranfang ---------
;
9dce 20fb9d :6810 -start         jsr parameter+3;spei-
9dd1 8de299 :6820 -              sta spanf       ;cheranf.
9dd4 8ce399 :6830 -              sty spanf+1     ;festlegen
9dd7 4cf799 :6840 -              jmp init
;
;-------- bild/farbspeicher ------
;
9dda 20fb9d :6880 -screen        jsr parameter+3;anfang
9ddd 8dd399 :6890 -              sta bsconst     ;bildsch.
9de0 8cd499 :6900 -              sty bsconst+1
9de3 60     :6910 -              rts             ;basic
;
;------ bildschirmfarben --------
;
9de4 209eb7 :6950 -color         jsr getbyte
9de7 8ec799 :6960 -              stx flag2
9dea 60     :6970 -              rts
;
;---- tastaturvector ein/aus ----
;
9deb 209eb7 :7010 -tast          jsr getbyte
9dee e000   :7020 -              cpx #0
9df0 d003   :7030 -              bne tast1
9df2 4c4f9a :7040 -              jmp vecoff
9df5 4c449a :7050 -tast1         jmp vecon
;
;
9df8 20fdae :7080 -parameter jsr chkcom
9dfb 208aad :7090 -              jsr frmnum
9dfe 20f7b7 :7100 -              jsr integer
9e01 a514   :7110 -              lda $14
9e03 a415   :7120 -              ldy $15
9e05 60     :7130 -              rts
;
;--------------------------------
;--        texte speichern     --
;--------------------------------
```

Listing 4/6.3.9.3-2 (Teil 14)

```
;
9e06 adce99 :7190 -speicher  lda namec
9e09 d009   :7200 -          bne speichera
9e0b a9e5   :7210 -          lda #<(errmsg) ;fehler-
9e0d 8522   :7220 -          sta $22        ;meldung
9e0f a999   :7230 -          lda #>(errmsg) ;ausgeben
9e11 4c45a4 :7240 -          jmp errout
;
9e14 2057e2 :7260 -speichera jsr $e257      ;name
;
9e17 20f89d :7280 -          jsr parameter  ;spaetere
9e1a 85f9   :7290 -          sta sp         ;start-
9e1c 84fa   :7300 -          sty sp+1       ;adresse
;
9e1e adce99 :7320 -          lda namec      ;anzahl
9e21 48     :7330 -          pha            ;retten
9e22 a900   :7340 -          lda #0
9e24 8dcf99 :7350 -          sta namec+1
9e27 0ece99 :7360 -          asl namec      ;anzahl
9e2a 2ecf99 :7370 -          rol namec+1    ;*2
9e2d 18     :7380 -          clc
9e2e adce99 :7390 -          lda namec      ;anz. +2 !
9e31 6902   :7400 -          adc #2
9e33 9003   :7410 -          bcc speicher8
9e35 eecf99 :7420 -          inc namec+1
9e38 8dce99 :7430 -speicher8 sta namec
9e3b 18     :7440 -          clc
;
9e3c a5f9   :7460 -          lda sp         ;wert zum
9e3e 6dce99 :7470 -          adc namec      ;speicher-
9e41 8dde99 :7480 -          sta spnew      ;anfang
9e44 a5fa   :7490 -          lda sp+1       ;addieren
9e46 6dcf99 :7500 -          adc namec+1
9e49 8ddf99 :7510 -          sta spnew+1
9e4c 18     :7520 -          clc
;
9e4d a902   :7540 -          lda #2         ;filenr.
9e4f a208   :7550 -          ldx #8         ;8=floppy
9e51 a002   :7560 -          ldy #2         ;sek.adr
9e53 20baff :7570 -          jsr setfls
;
9e56 20c0ff :7590 -          jsr open       ;file
9e59 a202   :7600 -          ldx #2         ;oeffnen
9e5b 20c9ff :7610 -          jsr ckout
;
9e5e a590   :7630 -          lda $90        ;fehler ?
9e60 f003   :7640 -          beq speichl1
9e62 4c129f :7650 -          jmp speicher5
;
9e65 a954   :7670 -speichl1  lda #"t"       ;erkenn-
9e67 20d2ff :7680 -          jsr bsout      ;ungsmarke
```

Listing 4/6.3.9.3-2 (Teil 15)

```
9e6a a590   :7690 -           lda $90        ;('tcs')
9e6c f003   :7700 -           beq speich10   ;ausgeben
9e6e 4c129f :7710 -           jmp speicher5
9e71 a943   :7720 -speich10   lda #"c"
9e73 20d2ff :7730 -           jsr bsout
9e76 a953   :7740 -           lda #"s"
9e78 20d2ff :7750 -           jsr bsout
;
9e7b a5f9   :7770 -speicher9  lda sp         ;startadr.
9e7d 20d2ff :7780 -           jsr bsout      ;speichern
9e80 a5fa   :7790 -           lda sp+1
9e82 20d2ff :7800 -           jsr bsout
;
9e85 68     :7820 -           pla            ;anzahl bs
9e86 8dce99 :7830 -           sta namec
9e89 20d2ff :7840 -           jsr bsout      ;speichern
;
9e8c add899 :7860 -           lda tbalt      ;anfang
9e8f acd999 :7870 -           ldy tbalt+1    ;sprung-
9e92 85f9   :7880 -           sta sp         ;tabelle
9e94 84fa   :7890 -           sty sp+1
9e96 a000   :7900 -           ldy #0
;
9e98 adde99 :7920 -           lda spnew      ;die
9e9b 20d2ff :7930 -           jsr bsout      ;ersten 2
9e9e addf99 :7940 -           lda spnew+1    ;bytes
9ea1 20d2ff :7950 -           jsr bsout      ;der
9ea4 a590   :7960 -           lda $90        ;sprungtab
9ea6 d06a   :7970 -           bne speicher5  ;speichern
9ea8 20439f :7980 -           jsr spinc
9eab 20439f :7990 -           jsr spinc
;
9eae 202c9f :8010 -speicher7  jsr read       ;relative
9eb1 85fd   :8020 -           sta tb         ;anfangs-
9eb3 20439f :8030 -           jsr spinc      ;adresse
9eb6 202c9f :8040 -           jsr read       ;holen
9eb9 85fe   :8050 -           sta tb+1
9ebb 20439f :8060 -           jsr spinc
;
9ebe a5fd   :8080 -           lda tb         ;beide
9ec0 d00d   :8090 -           bne speicher6  ;bytes =0?
9ec2 a5fe   :8100 -           lda tb+1
9ec4 d009   :8110 -           bne speicher6
9ec6 20d2ff :8120 -           jsr bsout      ;ja, ende
9ec9 20d2ff :8130 -           jsr bsout      ;der tab.
9ecc 4cf09e :8140 -           jmp speicher3  ;weiter
;
9ecf a5fd   :8160 -speicher6  lda tb         ;sprungadr
9ed1 18     :8170 -           clc
9ed2 6dde99 :8180 -           adc spnew      ;zur an-
9ed5 85fd   :8190 -           sta tb         ;fangsadr.
```

Listing 4/6.3.9.3-2 (Teil 16)

```
9ed7 a5fe    :8200 -             lda tb+1        ;addieren
9ed9 6ddf99  :8210 -             adc spnew+1
9edc 85fe    :8220 -             sta tb+1
9ede 18      :8230 -             clc
;
9edf a5fd    :8250 -             lda tb          ;und in
9ee1 20d2ff  :8260 -             jsr bsout       ;der
9ee4 a5fe    :8270 -             lda tb+1        ;tabelle
9ee6 20d2ff  :8280 -             jsr bsout       ;speichern
9ee9 a590    :8290 -             lda $90
9eeb d025    :8300 -             bne speicher5
9eed 4cae9e  :8310 -             jmp speicher7
;
9ef0 adda99  :8330 -speicher3 lda spalt
9ef3 acdb99  :8340 -             ldy spalt+1     ;text-
9ef6 85f9    :8350 -             sta sp          ;speicher
9ef8 84fa    :8360 -             sty sp+1
;
9efa 201b9f  :8380 -speicher4 jsr speichern   ;byte
9efd 20439f  :8390 -             jsr spinc       ;speichern
9f00 a590    :8400 -             lda $90         ;fehler ?
9f02 d00e    :8410 -             bne speicher5
;
9f04 a5f9    :8430 -             lda sp          ;letztes
9f06 cdd199  :8440 -             cmp spconst     ;byte ?
9f09 d0ef    :8450 -             bne speicher4
9f0b a5fa    :8460 -             lda sp+1
9f0d cdd299  :8470 -             cmp spconst+1
9f10 d0e8    :8480 -             bne speicher4
;
9f12 20ccff  :8500 -speicher5 jsr clrch       ;file
9f15 a902    :8510 -             lda #2          ;schliess-
9f17 20c3ff  :8520 -             jsr close       ;en
9f1a 60      :8530 -             rts             ;basic
;
;------ unterprogramme ----------
;
9f1b 48      :8570 -speichern pha             ;register
9f1c 8a      :8580 -             txa             ;retten
9f1d 48      :8590 -             pha
9f1e 98      :8600 -             tya
9f1f 48      :8610 -             pha
;
9f20 202c9f  :8630 -             jsr read        ;byte
9f23 20d2ff  :8640 -             jsr bsout       ;speichern
;
9f26 68      :8660 -             pla             ;register
9f27 a8      :8670 -             tay             ;zurueck
9f28 68      :8680 -             pla
9f29 aa      :8690 -             tax
9f2a 68      :8700 -             pla
```

Listing 4/6.3.9.3-2 (Teil 17)

```
9f2b 60      :8710 -           rts
;
;
9f2c a501    :8740 -read       lda 1           ;speicher
9f2e 8dd799  :8750 -           sta konf        ;auf ram
9f31 29f8    :8760 -           and #248        ;schalten
9f33 78      :8770 -           sei
9f34 8501    :8780 -           sta 1
9f36 a000    :8790 -           ldy #0
9f38 b1f9    :8800 -           lda (sp),y      ;wert
9f3a acd799  :8810 -           ldy konf        ;holen
9f3d 8401    :8820 -           sty 1
9f3f 58      :8830 -           cli             ;speicher
9f40 a000    :8840 -           ldy #0          ;zureuck
9f42 60      :8850 -           rts             ;weiter
;
;-------- sp erhoehen -----------
;
9f43 e6f9    :8890 -spinc      inc sp
9f45 d002    :8900 -           bne spinc1
9f47 e6fa    :8910 -           inc sp+1
9f49 18      :8920 -spinc1     clc
9f4a 60      :8930 -           rts
;
;---------------------------------
;--erweiterung der interpreters.-
;---------------------------------
;
;------ befehlsauswertung -------
;
9f4b 207300  :9010 -inter      jsr chrget      ;zeichen
9f4e c960    :9020 -           cmp #$60        ;holen
9f50 b019    :9030 -           bcs interout    ;kein
9f52 c941    :9040 -           cmp #$41        ;buch-
9f54 9015    :9050 -           bcc interout    ;stabe ?
9f56 8dcd99  :9060 -           sta akku        ;zeichen
9f59 a200    :9070 -           ldx #00         ;sichern
9f5b 8ee499  :9080 -           stx befnr       ;zaehler=0
9f5e a000    :9090 -inter1     ldy #00
9f60 eee499  :9100 -           inc befnr       ;befehls-
9f63 bdcd9f  :9110 -           lda befehle,x   ;tabelle
9f66 d009    :9120 -           bne inter2      ;auslesen
;
9f68 adcd99  :9140 -           lda akku        ;zurueck
9f6b 207900  :9150 -interout   jsr chrgot      ;zum
9f6e 4ce7a7  :9160 -           jmp intend      ;BASIC
;
;------ adresse suchen ----------
;
9f71 d17a    :9200 -inter2     cmp (txtptr),y  ;basicbef
9f73 d035    :9210 -           bne rest        ;=tabelle?
```

Listing 4/6.3.9.3-2 (Teil 18)

```
9f75 e8      :9220 -             inx              ;zaehler+1
9f76 c8      :9230 -             iny
9f77 bdcd9f  :9240 -             lda befehle,x    ;letztes
9f7a c980    :9250 -             cmp #128         ;zeichen ?
9f7c 90f3    :9260 -             bcc inter2
9f7e 297f    :9270 -             and #127         ;bit 7
9f80 d17a    :9280 -             cmp (txtptr),y   ;loeschen
9f82 f004    :9290 -             beq inter3       ;auch ok ?
9f84 e8      :9300 -             inx              ;nein,
9f85 4c5e9f  :9310 -             jmp inter1       ;weiter
;
9f88 18      :9330 -inter3       clc              ;txtptr
9f89 c8      :9340 -             iny              ;auf
9f8a 98      :9350 -             tya              ;aktuellen
9f8b 657a    :9360 -             adc txtptr       ;stand
9f8d 857a    :9370 -             sta txtptr       ;bringen
9f8f 9002    :9380 -             bcc lab1
9f91 e67b    :9390 -             inc txtptr+1
;
9f93 ade499  :9410 -lab1         lda befnr        ;befnr *2
9f96 0a      :9420 -             asl
9f97 aa      :9430 -             tax
9f98 bdb89f  :9440 -             lda befadr,x     ;sprung-
9f9b bcb99f  :9450 -             ldy befadr1,x    ;tabelle
9f9e 8da59f  :9460 -             sta jump+1       ;auslesen
9fa1 8ca69f  :9470 -             sty jump+2       ;und neue
9fa4 20ffff  :9480 -jump         jsr $ffff        ;routine
9fa7 4c6b9f  :9490 -             jmp interout
;
;---- befehlstext ueberlesen ----
;
9faa e8      :9530 -rest         inx              ;zaehler+1
9fab bdcd9f  :9540 -             lda befehle,x    ;letztes
9fae 2980    :9550 -             and #128         ;zeichen
9fb0 c900    :9560 -             cmp #0           ;der tab.?
9fb2 f0f6    :9570 -             beq rest         ;nein.
9fb4 e8      :9580 -             inx              ;ja,
9fb5 4c5e9f  :9590 -             jmp inter1       ;weiter
;
;
;--------------------------------
;-- sprungtabelle neue befehle --
;--------------------------------
;
              9660 -befadr       .by $e7
              9670 -befadr1      .by $a7
              9680 -             .wo init
              9690 -             .wo speicher
              9700 -             .wo start
              9710 -             .wo screen
              9720 -             .wo color
```

Listing 4/6.3.9.3-2 (Teil 19)

```
                9730 -          .wo anz
                9740 -          .wo mem
                9750 -          .wo tast
                9760 -          .wo interoff
                9770 -          .by 0
;
;--------------------------------
;--        befehlstabelle      --
;--------------------------------
;
                9830 -befehle   .tx "iniT"
                9840 -          .tx "disK"
                9850 -          .tx "starT"
                9860 -          .tx "screeN"
                9870 -          .tx "coL"
                9880 -          .tx "nuM"
                9890 -          .tx "meM"
                9900 -          .tx "tasT"
                9910 -          .tx "ofF"
                9920 -          .by 0
```

Listing 4/6.3.9.3-2 (Teil 20)

4/6.4

Basic-Erweiterungen beim C 64

(Autor: Manfred Friese)

Das Basic des Commodore 64 kennt leider nur eine sehr begrenzte Anzahl von Befehlen. Daher kann man seine Leistungsfähigkeit nur mit Hilfe von POKE und PEEK-Befehlen oder mit Hilfe von Maschinenprogrammen richtig nutzen. Diesen Nachteil haben zahlreiche Softwarefirmen erkannt und bieten Basicerweiterungen für den C 64 an, die mit Hilfe von neuen Befehlen die Hardware besser nutzen sollen. Simon's Basic und Extended Basic sind die wohl bekanntesten Befehlserweiterungen. Doch eine solche Erweiterung kann man auch selbst schreiben, wie dieser Beitrag zeigen wird.

4/6.4.1

Allgemeine Vorgehensweise

Wir wollen zunächst ein möglichst günstiges Konzept für eine Basic-Erweiterung erarbeiten. Dazu werden wir uns eine Reihe von Symbolen zur Verfügung stellen, welche die wichtigsten Adressen im Betriebssystem des Commodore 64 bezeichnen, aber auch die Nutzung des Speichers durch unsere Erweiterung festlegen. Anschließend betrachten wir die Initialisierung unseres Systems und die Arbeitsweise der Befehlserweiterung.

Allen folgenden Kapiteln ist das Protokoll eines Assemblerlaufs mit der Basic-Erweiterung angegliedert. Die Kapitel enthalten nur die zum jeweiligen Thema gehörenden Zeilen. Insgesamt ergeben die Zeilen eine lauffähige Basic-Erweiterung (was Sie auch an der Zeilennumerierung ersehen können), welche in den folgenden Ergänzungen noch weiter ausgebaut werden wird. Jedoch bietet Ihnen die Grundversion bereits eine Reihe nützlicher Befehle.

4/6.4.1.1

Symbolvereinbarungen

Im Folgenden finden Sie die Tabelle der wichtigsten Symbole, die wir für unsere Basic-Erweiterung benötigen:

```
;;;XBASIC VERSION 1.0  (c) M.Friese 5/87

;--------------------------------------
;---     Vereinbarung der Symbole    ---
;--------------------------------------
xtoken:     .eq 210             ;Tokennummer der Erweiterung

pp:         .eq 1               ;Prozessorport
xmem:       .eq 2               ;X-Register fuer ROM-Routinen
azg:        .eq 251             ;allgemeiner Zeiger 1
azg2:       .eq 253             ;allgemeiner Zeiger 2
basend:     .eq $37             ;Basic Speicherende
charpage:   .eq 648             ;Festlegung der Textseite
nmivec:     .eq $318            ;NMI-Vektor
listvec:    .eq $306            ;LIST Vektor
tokenvec:   .eq $304            ;In Token wandeln
executevec:.eq $308             ;Ausfuehrungsvektor
fktvec:     .eq $30a            ;Funtionen Vektor
zeile:      .eq $200            ;Adresse der Eingabezeile
mem:        .eq 828             ;allgemeiner Puffer

vic:        .eq $d000           ;Video IC
sid:        .eq $d400           ;Sound IC
farbram:    .eq $d800           ;Farbram
cia1:       .eq $dc00           ;CIA #1
cia2:       .eq $dd00           ;CIA #2
zeichensatz:.eq $d000           ;Startadresse Zeichensatz

xbasicstart:.eq $a000-1024-512;Startadresse von XBASIC
textseite: .eq $cc00            ;Adresse der Textseite
farbehires:.eq $c800            ;Adresse der Farbe fuer Hires
hiresseite:.eq $e000            ;Adresse der Hiresseite

chrget:     .eq $73             ;ein Zeichen aus Basic holen
chrgot:     .eq $79             ;Zeichen noch einmal lesen
varsuch:    .eq $b08b           ;Variale suchen
```

Listing 4/6.4.1.1 (Teil 1)

```
bsout:      .eq $ffd2          ;ein Zeichen ausgeben
cursor:     .eq $fff0          ;Cursor setzen/loeschen
CURrechts:  .eq $ab3b          ;Cursor rechts
reset:      .eq $fce2          ;Computer Reset
ROMfrmevl:  .eq $ad9e          ;Ausdruck berechnen
ROMchkklazu:.eq $aef7          ;auf 'Klammer zu' testen
ROMchklaauf:.eq $aefa          ;auf 'Klammer auf' testen
ROMklammer:.eq $aef1           ;Klammerausdruck auswerten
ROMadresse:.eq $b7f7           ;FAC->pos. Integer
testcode:   .eq $aeff          ;auf Code testen
frestr:     .eq $b6a6          ;String aufraeumen
garbage:    .eq $b526          ;Garbarge collection
ROMresstr:  .eq $b47d          ;String reservieren
ROMstring:  .eq $b4ca          ;String in Stringstack
getstring:  .eq $b782          ;String holen
clrscr:     .eq $e544          ;Bildschirm loeschen
get:        .eq $ffe4          ;GET-Befehl
ROMgetbyte:.eq $b79e           ;Holt ein Byte
ROMchkkom:  .eq $aefd          ;auf Komma pruefen

initmeldung:.eq $e42d          ;Startmeldung ausgeben
basicwasta:.eq $e386           ;Basic Warmstart
```

Listing 4/6.4.1.1 (Teil 2)

Neben den Adressen der wichtigen ROM-Routinen enthält diese Aufstellung eine Reihe von Vektoren, über die wir unsere Basic-Erweiterung in das normale Basic einbauen werden. Bei diesen Vektoren handelt es sich um Adressen, welche in der Zero-Page des Commodore 64 stehen, die auf wichtige Routinen des Basic-Interpreters zeigen. Wird z.B. der LIST-Befehl ausgeführt, so wird die entsprechende Routine im ROM nicht direkt angesprungen, sondern über den Befehl JMP($306) aufgerufen. Daher muß in den Speicherstellen $306 und $307 die Adresse der LIST-Routine stehen. Schreiben wir eine andere Adresse in diese Speicherstellen, können wir eine eigene LIST-Routine verwenden.

Außerdem enthält die Symbolvereinbarung noch die Adressen, welche die Speicherverwaltung für unsere Basic-Erweiterung festlegen. Der Erweiterung wurde folgender Speicheraufbau zugrunde gelegt:

Adresse	Normal		Erweiterung		Symbol
	Kernal		Kernal	Graphic	hires-seite
$E000	I/O	Zeichensatz im ROM	I/O	Zeichensatz im RAM	zeichensatz
$D000 $CC00 $C800 $C400	RAM		Textseite Graphic Farbe max. 16 Sprites Frei		textseite farbehires
$C000	BASIC ROM		BASIC ROM	BASIC Erweiterung	
$A000 $9A00	RAM für BASIC		RAM für BASIC		xbasicstart
$8000					

Wie Sie der Übersicht entnehmen können, benötigt unsere Basic-Erweiterung nur 1,5 KByte des Basic-Speicherplatzes. In diesen 1,5 KByte liegen die Systemroutinen der Erweiterung. Die Routinen der einzelnen Befehle liegen unter dem Basic-ROM. Dies wird uns jedoch nicht daran hindern, Routinen aus dem Basic-ROM für unsere Befehle zu verwenden.

Der Speicherbereich des Video-IC's wurde in den Bereich von $C000-$FFFF verlegt. Dies hat den Nachteil, daß Programme, die den Bildschirmbereich direkt mit POKE beschreiben, nicht mit der Basic-Erweiterung zusammenarbeiten. Dem gegenüber stehen aber mehrere Vorteile. Der Zeichensatz kann in das unter dem I/O-Bereich liegende RAM untergebracht werden. So ist es möglich eigene Zeichen zu definieren (z.B. deutsche Umlaute). Eine Graphik-Seite kann unter dem Kernal angelegt werden, und im Bereich von $C400-$C7FF können bis zu 16 Spritedefinitionen abgelegt werden, die sowohl für die Textseite, als auch für die Graphikseite gelten. Der Speicherbereich von $C000-$C3FF bleibt vorerst frei, da viele Maschinenprogramme, die Sie vielleicht noch nutzen möchten, diesen Bereich belegen.

Da die Textseite nach $CC00 verlegt wurde, wird der Bereich von $400-$7FF, in der die Textseite normalerweise liegt, frei. Diesen könnte man nun für Basicprogramme

nutzen, so daß für Basicprogramme nur noch 0,5 KByte durch die Basic-Erweiterung verloren gingen. Dies werden wir jedoch nicht tun, daß viele Programme die normale Startadresse erwarten (z.B. Compiler, compilierte Programme und Programme mit Maschinensprache). Uns steht dieser Bereich für eigene Zwecke zur Verfügung. Die Grundversion der Basic-Erweiterung macht jedoch noch keinen Gebrauch von diesem Bereich.

4/6.4.1.2

Initialisierung

Die Basic-Erweiterung soll als normales Programm von Diskette gelesen und dann mit RUN gestartet werden. Um dies zu erreichen, muß der Assembler als Startadresse die normale Startadresse von Basicprogrammen verwenden. Ab dieser Adresse wird eine Basic-Zeile erzeugt. Sie startet das Maschinenprogramm, welches direkt an die Basic-Zeile anschließt. Die Basic-Zeile muß daher '1987 SYS 2061' lauten, wobei sich die Startadresse des Maschinenprogramms (2061) aus der Startadresse der Basic-Zeile und ihrer Länge ergibt. Die Zeilennummer ist frei gewählt, und soll hier für das Jahr der Entwicklung des Programms stehen. Das so gestartete Maschinenprogramm erledigt nun nicht etwa die Einbindung der Erweiterung in den Basic-Interpreter des Commodore 64, sondern verschiebt die Erweiterung an seine Startadresse. Dann wird die Basicerweiterung an ihrer Startadresse gestartet.

Bei der Assemblierung muß die eigentliche Erweiterung natürlich der Verschieberoutine folgen, aber zu ihrer Startadresse assembliert werden. Unser Assembler stellt diese Möglichkeit mit dem '.cb'-Befehl zur Verfügung.

Die Erweiterung bindet sich dann in den Basic-Interpreter ein. Zunächst wird der Zeichensatz von dem ROM-Bereich in den RAM-Bereich übertragen. Dies leistet die Routine <setzeichen>. Dann werden die Vektoren des Basic-Interpreters auf die Erweiterung umgeleitet. Außerdem wird der Zugriffsbereich des Video-IC's in den oberen 16 KByte-Bereich verlegt.

Damit die Basic-Erweiterung nicht von Zeichenketten oder Programmzeilen überschrieben werden kann, muß der Basic-Bereich noch um 1,5 KByte eingeschränkt werden. Die Initialisierung ist dann mit der Ausgabe der Einschaltmeldung abgeschlossen.

```
                  ;---------------------------------------
                  ;---   XBASIC Basic-Startroutine     ---
                  ;---------------------------------------
                            .wa                 ;bei Fehler warten
0801                        .ba $801            ;Basic Start
                            .ou "@0:xbasic"     ;erzeugte Datei auf Disk schre
iben
                            .oc                 ;keine Ausgabe in das RAM
0801 0b 08 c3               .by 11,8,<1987,>1987,158,"2061",0,0,0;Entspricht
"1987 SYS2061"
07 9e 32 30 36 31 00 00 00
080d a9 32                  lda #<XBASIC        ;An Normalstart schieben
080f a2 08                  ldx #>XBASIC        ;Zeiger setzen
0811 85 fb                  sta azg
0813 86 fc                  stx azg+1
0815 a9 00                  lda #<xbasicstart   ;Normaladresse
0817 a2 9a                  ldx #>xbasicstart
0819 85 fd                  sta azg2
081b 86 fe                  stx azg2+1
081d a2 24                  ldx #4*9            ;9 KByte uebertragen
081f a0 00                  ldy #0
0821 b1 fb        TRABASIC: lda (azg),y         ;Verschiebeschleife
0823 91 fd                  sta (azg2),y
0825 c8                     iny
0826 d0 f9                  bne TRABASIC        ;bis eine Page verschoben
0828 e6 fc                  inc azg+1
082a e6 fe                  inc azg2+1
082c ca                     dex
082d d0 f2                  bne TRABASIC        ;bis alle Seiten verschoben
082f 4c 00 9a               jmp xbasicstart     ;XBASIC starten
                  XBASIC:
                            .cb xbasicstart     ;Rest zur Normaladresse assemb
lieren
                  ;---------------------------------------
                  ;---  XBASIC Initialisierungsroutine ---
                  ;---------------------------------------
9a00 a9 37        BBxbasic: lda #$37            ;Speicheraufteilung normal
9a02 85 01                  sta pp
9a04 20 64 9a               jsr setzeichen
9a07 20 17 9a               jsr initxbasic      ;XBASIC Vektoren setzen
9a0a a9 e2                  lda #<meldung       ;Adresse der Einschaltmeldung
laden
9a0c a0 9d                  ldy #>meldung
9a0e 20 2d e4               jsr initmeldung     ;Ausgabe im Commodore 64 Basic
9a11 a2 fb                  ldx #$fb            ;Stackpointer laden
9a13 9a                     txs
9a14 4c 86 e3               jmp basicwasta      ;und zum Basic-Warmstart
9a17 a9 36        initxbasic:lda #%00110110     ;Adresse des Bildschirms
9a19 8d 18 d0               sta vic+24          ;und des Zeichensatzes festleg
en
```

Listing 4/6.4.1.2 (Teil 1)

```
9a1c ad 00 dd                  lda cia2           ;VIC Bereich von $C000-$FFFF
9a1f 29 fc                     and #%11111100
9a21 8d 00 dd                  sta cia2
9a24 a9 cc                     lda #textseite/256 ;Textseite fuer Basic festlege
n
9a26 8d 88 02                  sta charpage

9a29 a9 00                     lda #<xbasicstart  ;Neues Basic-Ende setzen
9a2b a2 9a                     ldx #>xbasicstart
9a2d 85 37                     sta basend
9a2f 86 38                     stx basend+1
9a31 a9 87                     lda #<newnmi       ;neue NMI-Routine
9a33 a2 9a                     ldx #>newnmi
9a35 8d 18 03                  sta nmivec
9a38 8e 19 03                  stx nmivec+1
9a3b a9 99                     lda #<token        ;neue Token-Wandelroutine
9a3d a2 9b                     ldx #>token
9a3f 8d 04 03                  sta tokenvec
9a42 8e 05 03                  stx tokenvec+1
9a45 a9 84                     lda #<list         ;neue List-Routine
9a47 a2 9d                     ldx #>list
9a49 8d 06 03                  sta listvec
9a4c 8e 07 03                  stx listvec+1
9a4f a9 36                     lda #<execute      ;neue Ausfuehrungsroutine
9a51 a2 9e                     ldx #>execute
9a53 8d 08 03                  sta executevec
9a56 8e 09 03                  stx executevec+1
9a59 a9 0a                     lda #<exefkt       ;Neue Funktionenroutine
9a5b a2 9e                     ldx #>exefkt
9a5d 8d 0a 03                  sta fktvec
9a60 8e 0b 03                  stx fktvec+1
9a63 60                        rts

9a64 78             setzeichen:sei
9a65 a5 01                     lda pp
9a67 48                        pha
9a68 a2 10                     ldx #4*4           ;4 KByte Zeichensatz von ROM i
n RAM bringen
9a6a a9 d0                     lda #>zeichensatz
9a6c a0 00                     ldy #0
9a6e 85 fc                     sta azg+1          ;azg zeigt auf Zeichensatz
9a70 84 fb                     sty azg
9a72 a9 32                     lda #$32           ;Zeichensatz-ROM lesbar machen
9a74 85 01                     sta pp
9a76 b1 fb          trazeichen:lda (azg),y        ;aus ROM in RAM bringen
9a78 91 fb                     sta (azg),y
9a7a c8                        iny                ;und weiter
9a7b d0 f9                     bne trazeichen
9a7d e6 fc                     inc azg+1
9a7f ca                        dex                ;bis alle Seiten uebertragen
9a80 d0 f4                     bne trazeichen
9a82 68                        pla                ;normale Speicherbelegung
9a83 85 01                     sta pp
9a85 58                        cli
9a86 60                        rts
```

Listing 4/6.4.1.2 (Teil 2)

4/6.4.1.3

Eine neue NMI-Routine

Bei einem Druck auf die RESTORE-Taste wird dem Prozessor eine Interrupt-Anforderung gemeldet. Der Basic-Interpreter initialisiert daraufhin einige Vektoren und Speicherstellen neu. Dies würde unsere Basic-Erweiterung sofort wieder aus dem System auskoppeln. Um dies zu verhindern, verwenden wir eine eigene NMI-Routine.

Unsere NMI-Routine besteht im wesentlichen aus der Kopie der NMI-Routine des normalen Basic-Interpreters. Wir binden lediglich die Routine <initxbasic>, welche die Vektoren für unsere Basic-Erweiterung setzt, in die NMI-Routine ein. Die Speicherstellen werden dann zwar von den Routinen des Interpreters auf die alten Werte gesetzt, aber sofort durch unsere Routine restauriert. Die Erweiterung bleibt uns so erhalten.

```
                  ;--------------------------------------
                  ;---  XBASIC NMI Routine            ---
                  ;--------------------------------------
9a87 48           newnmi:   pha                ;Kopie der Commodore 64 NMI-Routine
9a88 8a                     txa
9a89 48                     pha
9a8a 98                     tya
9a8b 48                     pha
9a8c a9 7f                  lda #$7f
9a8e 8d 0d dd               sta $dd0d
9a91 ac 0d dd               ldy $dd0d
9a94 30 1d                  bmi rs232
9a96 20 bc f6               jsr $f6bc
9a99 20 e1 ff               jsr $ffe1
9a9c d0 15                  bne rs232
9a9e a2 fb                  ldx #$fb
9aa0 9a                     txs
9aa1 20 15 fd               jsr $fd15
9aa4 20 17 9a               jsr initxbasic     ;XBASIC einbinden
9aa7 20 a3 fd               jsr $fda3
9aaa 20 18 e5               jsr $e518
9aad 20 17 9a               jsr initxbasic     ;XBASIC einbinden
9ab0 6c 02 a0               jmp ($a002)        ;weiter im Commodore 64 Basic

9ab3 4c 72 fe     rs232:    jmp $fe72
```

Listing 4/6.4.1.3

4/4.6.1.4

Nutzen der Basic-ROM-Routinen

Wir haben bereits festgestellt, daß die Routinen der Befehle unserer Basic-Erweiterung im RAM unter dem Basic-Interpreter liegen. Um diese Routinen ausführen zu können, muß der Basic-Interpreter also abgeschaltet werden. Trotzdem wollen wir die im Basic-Interpreter enthaltenen Routinen für unsere Zwecke nutzen.

Um dies zu ermöglichen, muß der Basic-Interpreter für die Abarbeitung einer Routine wieder eingeschaltet werden. Wir legen uns also eine Reihe von Routinen im 1,5 KByte großen RAM-Bereich vor dem Basic-ROM an, welche den Basic-Interpreter einschalten, eine bestimmte Routine ausführen, und anschließend den Basic-Interpreter wieder abschalten. Diese Methode wurde bei der Modulversion des Assemblers aus diesem Buch bereits erfolgreich angewendet.

Leider ist es ziemlich aufwendig für alle benötigten Routinen des Basic-Interpreters eine eigene Routine im RAM-Bereich anzulegen. Andererseits sollen die Routinen auch nicht neu geschrieben werden. Wir legen daher eine Tabelle an, in der die Adressen von weiteren, benötigten Routinen abgelegt werden. Diese Routinen können dann über eine Nummer aufgerufen werden. Dazu wird das X-Register mit der Nummer der Routine geladen und die Routine <basicrom> aufgerufen. Um dem X-Register trotzdem bei dem Start der Routine einen definierten Wert zuzuweisen, kann in der Speicherstelle 'xmem' der Wert für das X-Register abgelegt werden.

Im Anschluß an die Tabelle wird ein Speicherbereich für Erweiterungen freigehalten.

```
                   ;-------------------------------------
                   ;---  BASIC ROM-Routinen aufrufen   ---
                   ;-------------------------------------
9ab6 8d 66 9b      basicrom:  sta akku           ;Akku merken
9ab9 20 c5 9a                 jsr romaufruf      ;Routine aufrufen
9abc 08                       php                ;Staus merken
9abd 48                       pha                ;Akku merken
9abe a9 36                    lda #$36           ;ROM ausschalten
9ac0 85 01                    sta pp
9ac2 68                       pla                ;Akku wiederherstellen
9ac3 28                       plp                ;Status zurueckholen
9ac4 60                       rts

9ac5 bd 68 9b      romaufruf: lda rombefehle+1,x ;Adresse der Routine auf den S
tack bringen
```

Listing 4/6.4.1.4 (Teil 1)

```
9ac8 48                       pha
9ac9 bd 67 9b                 lda rombefehle,x
9acc 48                       pha
9acd a9 37                    lda #$37          ;Basicrom einschalten
9acf 85 01                    sta pp
9ad1 ad 66 9b                 lda akku          ;Akku laden
9ad4 a6 02                    ldx xmem          ;X-Register laden
9ad6 60                       rts
9ad7 e6 01        getazg:     inc pp            ;Wert unter azg aus ROM holen
9ad9 b1 fb                    lda (azg),y
9adb c6 01                    dec pp
9add 60                       rts
9ade 86 22        adrfehler:  stx $22           ;Fehlermeldung ab Adresse A/X
9ae0 e6 01                    inc pp            ;ausgeben
9ae2 4c 45 a4                 jmp $a445
9ae5 57 52 4f     slerrtxt:   .by "wrong stringleN"
4e 47 20 53 54 52 49 4e 47 4c 45 ce
9af4 42 41 44     bcerrtxt:   .by "bad char in strinG"
20 43 48 41 52 20 49 4e 20 53 54 52 49 4e c7
9b06 e6 01        fehler:     inc pp            ;Fehlermeldung mit Nummer X au
sgeben
9b08 4c 37 a4                 jmp $a437
9b0b e6 01        frmevl:     inc pp            ;Ausdruck einlesen
9b0d 20 9e ad                 jsr ROMfrmevl
9b10 c6 01                    dec pp
9b12 60                       rts
9b13 e6 01        chkklazu:   inc pp            ;Auf ')' pruefen
9b15 20 f7 ae                 jsr ROMchkklazu
9b18 c6 01                    dec pp
9b1a 60                       rts
9b1b e6 01        chklaauf:   inc pp            ;Auf '(' pruefen
9b1d 20 fa ae                 jsr ROMchklaauf
9b20 c6 01                    dec pp
9b22 60                       rts
9b23 e6 01        chkkom:     inc pp            ;Auf ',' pruefen
9b25 20 fd ae                 jsr ROMchkkom
9b28 c6 01                    dec pp
9b2a 60                       rts
9b2b e6 01        adresse:    inc pp            ;Adresse holen
9b2d 20 f7 b7                 jsr ROMadresse
9b30 c6 01                    dec pp
9b32 60                       rts
9b33 e6 01        getbyte:    inc pp            ;Bytewert holen
9b35 20 9e b7                 jsr ROMgetbyte
9b38 c6 01                    dec pp
```

Listing 4/6.4.1.4 (Teil 2)

```
9b3a 60                         rts

9b3b e6 01         getadr:      inc pp           ;16Bit-Wert holen
9b3d 20 8a ad                   jsr $ad8a
9b40 20 f7 b7                   jsr $b7f7
9b43 c6 01                      dec pp
9b45 60                         rts

9b46 e6 01         getadby:     inc pp           ;Adresse,Byte holen
9b48 20 eb b7                   jsr $b7eb
9b4b c6 01                      dec pp
9b4d 60                         rts

9b4e e6 01         reservstr:   inc pp           ;String reservieren
9b50 20 7d b4                   jsr ROMresstr
9b53 c6 01                      dec pp
9b55 60                         rts

9b56 e6 01         string:      inc pp           ;String eintragen
9b58 20 ca b4                   jsr ROMstring
9b5b c6 01                      dec pp
9b5d 60                         rts

9b5e e6 01         klammer:     inc pp           ;Klammerausdruck holen
9b60 20 f1 ae                   jsr ROMklammer
9b63 c6 01                      dec pp
9b65 60                         rts

9b66 00            akku:        .by 0            ;Speicher fuer Akku

                   rombefehle:
9b67 fe ae                      .wo testcode-1   ;0 <- Nummer
9b69 25 b5                      .wo garbage-1    ;2
9b6b 81 b7                      .wo getstring-1  ;4
9b6d f6 b8                      .wo $b8f7-1      ;6
9b6f 48 bc                      .wo $bc49-1      ;8
9b71 3a ab                      .wo CURrechts-1  ;10
9b73 e3 ff                      .wo get-1        ;12
9b75 6b e5                      .wo $e56c-1      ;14
9b77 a5 b6                      .wo frestr-1     ;16
9b79 32 a5                      .wo 42291-1      ;18
9b7b 8a b0                      .wo varsuch-1    ;20

                   RESERVE1:    .db 50+rombefehle-RESERVE1;Reserve fuer Erweiter
ungen

```

Listing 4/6.4.1.4 (Teil 3)

4/6.4.1.5

Wandeln in Token

Bevor eine Programmzeile in den Speicher abgelegt, oder eine Reihe von Direktbefehlen abgearbeitet werden kann, wird die Eingabezeile vom Basic-Interpreter in zwei Schritten bearbeitet:

1. Eine evtl. vorhandene Zeilennummer wird in eine 2 Byte-Binärdarstellung gewandelt.

2. Die Basic-Befehlsworte werden in Token gewandelt. Dabei wird jedes Wort in der Befehlstabelle gesucht. So kann jedem Wort genau eine Nummer zugeordnet werden. Zu dieser Nummer wird nun der Wert 128 addiert. Dieser 1 Byte-Wert heißt Token, und wird statt des Wortes in die Eingabezeile geschrieben.

Bei der in diesem Buch vorgestellten Basic-Erweiterung zum Commodore 128 wurde in diesen Mechanismus nicht eingegriffen, sondern die Befehle bei der Ausführung in der Zeile gesucht. Hier werden wir hingegen auch unsere Befehlsworte in Token umwandeln. Um nicht auf den Tokennummern des normalen Basic-Interpreters aufbauen zu müssen, werden wir als gemeinsamen Wert für alle Befehle den Wert 210 verwenden, der vom Commodore 64-Basic nicht verwendet wird. Diesem gemeinsamen Code folgt in einem weiteren Byte die Nummer des Befehls.

Unsere Routine <token> ruft zunächst die Originalroutine des Basic-Interpreters auf. Diese wandelt die Eingabezeile in Token, sowie eine evtl. vorhandene Zeilennummer in die interne Darstellung. Anschließend suchen wir in der Eingabezeile unsere neuen Befehlsworte. Stößt die Routine dabei auf eine Zeichenkette, so wird diese überlesen. Bei einem REM oder DATA-Befehl wird der Rest der Zeile nicht weiter beachtet. Alles andere wird mit den Befehlsworten in der Tabelle 'beflist' verglichen. Diese Tabelle muß sechs Bedingungen erfüllen:

1. Alle Funktionsnamen stehen vor den Befehlsnamen.

2. Enthält ein Befehl einen anderen als Teilnamen, so muß er vor diesem stehen. So muß z.B. INSTR vor INST stehen, da die Zeichenkette 'INST' ein Teil der Zeichenkette 'INSTR' ist.

3. Enthält ein Befehl den Namen eines normalen Basicbefehls, so muß der Name als Token angegeben werden. Bsp.: der Befehl DEFCHAR enthält als Teilnamen den Befehl DEF. Dieser wird von der Routine des Commodore 64 in das Token 150 gewandelt. Der Befehlsnahme für DEFCHAR muß daher als '150,"CHAR"' eingetragen werden.

4. Jeder Befehl enthält mindestens zwei Zeichen.
5. Jeder Befehl endet mit einem Nullbyte.
6. Die Tabelle endet mit einem Nullbyte.

Wird ein Befehlswort gefunden, überschreibt die Routine die ersten beiden Zeichen mit dem Wert 210 bzw. der Befehlsnummer. Der Rest des Wortes wird gelöscht. Nachdem die Zeile abgearbeitet ist, müssen noch die Bedingungen für die nachfolgenden Routinen im Basic-Interpreter hergestellt werden. Diese verlangen zum einen, daß das Y-Register die Länge der Zeile enthält, und zum anderen muß am Ende der Zeile eine Folge von drei Byte stehen, von denen das erste und das letzte ein Nullbyte ist. Das zweite Byte kann beliebig sein. Unsere Routine beachtet diese Bedingung.

```
                    ;---------------------------------------
                    ;---  Eingabe in Token wandeln       ---
                    ;---------------------------------------
9b99 20 7c a5       token:     jsr $a57c        ;Wandeln der normalen C64 Befe
hle
9b9c 84 0b                     sty $b           ;Zeilenlaenge merken
9b9e a0 ff                     ldy #$ff         ;Y auf die Eingabezeile
9ba0 84 fb                     sty azg          ;Y merken
9ba2 a2 ff          vgl1:      ldx #$ff         ;X zeigt auf die Befehlsliste
9ba4 a9 01                     lda #1           ;Tokennummer ist 1
9ba6 85 fd                     sta azg2
9ba8 c8             vgl:       iny              ;naechstes Zeichen vergleichen
9ba9 e8                        inx
9baa bd 20 9c                  lda beflist,x
9bad f0 2b                     beq eob
9baf d9 00 02                  cmp zeile,y
9bb2 d0 41                     bne ungl
9bb4 b9 00 02                  lda zeile,y
9bb7 c9 83                     cmp #131         ;DATA-Token ?
9bb9 f0 52                     beq aborttk      ;dann Rest der Zeile nicht bea
chten
9bbb c9 8f                     cmp #143         ;REM-Token ?
9bbd f0 4e                     beq aborttk      ;dann Rest nicht beachten
9bbf c9 22                     cmp #34          ;String ?
9bc1 d0 0d                     bne nohkomma     ;nein
9bc3 c8             hkomma:    iny              ;Strings ueberlesen
9bc4 b9 00 02                  lda zeile,y
9bc7 f0 44                     beq aborttk      ;Ende der Zeile erreicht
9bc9 c9 22                     cmp #34
9bcb d0 f6                     bne hkomma
9bcd 88                        dey
9bce d0 35                     bne nexttk       ;Stringende erreicht
9bd0 bd 20 9c       nohkomma:  lda beflist,x    ;Befehlstext mit Zeile vergl.
9bd3 d9 00 02                  cmp zeile,y
9bd6 d0 1d                     bne ungl         ;Zeichen ungleich
9bd8 f0 ce                     beq vgl          ;nein
9bda a6 fb          eob:       ldx azg          ;sonst Befehl gefunden
```

Listing 4/6.4.1.5 (Teil 1)

```
9bdc e8                     inx
9bdd a9 d2                  lda #xtoken        ;XBASIC-Token in Zeile bringen
9bdf 9d 00 02               sta zeile,x
9be2 e8                     inx
9be3 a5 fd                  lda azg2
9be5 9d 00 02               sta zeile,x
9be8 88                     dey
9be9 b9 01 02     versch:   lda zeile+1,y      ;Rest des Befehlswortes loesch
en
9bec 9d 01 02               sta zeile+1,x
9bef f0 24                  beq tonexttk       ;Laenge der Zeile korrigieren
9bf1 e8                     inx
9bf2 c8                     iny
9bf8 e6 fd                  inc azg2           ;Tokennummer +1
9bf3 d0 f4                  bne versch
9bf5 a4 fb        ungl:     ldy azg            ;Ende des Befehlswortes in Tab
elle suchen
9bf7 ca                     dex
9bfa e8           sbe:      inx
9bfb bd 20 9c               lda beflist,x
9bfe d0 fa                  bne sbe
9c00 bd 21 9c               lda beflist+1,x    ;Tabelle zuende?
9c03 d0 a3                  bne vgl            ;nein, weitervergleichen
9c05 c8           nexttk:   iny                ;ab naechte Position in Zeile
 neusuchen
9c06 84 fb                  sty azg
9c08 b9 01 02               lda zeile+1,y      ;Zeile beendet ?
9c0b d0 95                  bne vgl1           ;nein, weitersuchen
9c0d a4 0b        aborttk:  ldy $b             ;Laenge der Zeile laden
9c0f a9 00                  lda #0
9c11 99 fd 01               sta $200-3,y
9c14 60                     rts                ;fertig
9c15 8a           tonexttk: txa                ;Laenge der Zeile korrigieren
9c16 a4 fb                  ldy azg            ;Laenge der neuen Zeile
9c18 18                     clc                ;+6 fuer Verwaltung
9c19 69 06                  adc #6
9c1b 85 0b                  sta $b             ;ergibt die Laenge
9c1d d0 e6                  bne nexttk
                  ;---------------------------------------
                  ;---  Liste der XBASIC Befehle       ---
                  ;---------------------------------------
                  ;  1. Liste der Funktionen
9c1f 09           funktionen:.by 8+1           ;Anzahl der Funktionen +1
9c20 b8 28 00     beflist:  .by 184,'(,0       ;Liste der Funktionsworte
9c23 41 54 28               .by "at(",0
00
9c27 48 45 58               .by "hex$",0
24 00
9c2c 44 45 43               .by "dec",0
```

Listing 4/6.4.1.5 (Teil 2)

```
00
9c30 49 4e 4b          .by "inkey$",0
45 59 24 00
9c37 49 4e 53          .by "instr(",0
54 52 28 00
9c3e 53 54 52          .by "string$(",0
49 4e 47 24 28 00
9c47 53 50 41          .by "space$(",0
43 45 24 28 00

               ;  2. Liste der Befehle
9c4f 43 4c 53          .by "cls",0          ;Liste der Befehlsworte
00
9c53 4f 4c 44          .by "oldchar",0
43 48 41 52 00
9c5b 43 36 34          .by "c64",0
00
9c5f 58 42 41          .by "xbasic",0
53 49 43 00
9c66 49 4e 53          .by "inst",0
54 00
9c6b 96 43 48          .by 150,"char",0
41 52 00
9c71 4f 4c 44          .by "old",0
00
9c75 00                .by 0               ;Tabellenabschluss

               RESERVE2:   .db 256+beflist-RESERVE2;Reserve fuer Erweiterung
en


               ;Tabelle der Adressen der Befehle & Funktionen
9d20 f6 9f     befehle:    .wo BBfre-1         ;FRE(   * Funktionen
9d22 e4 a0                 .wo BBat-1          ;AT(
9d24 81 a0                 .wo BBhex-1         ;HEX$
9d26 29 a0                 .wo BBdez-1         ;DEC
9d28 ba a0                 .wo BBinkey-1       ;INKEY$
9d2a 98 a1                 .wo BBinstr-1       ;INSTR
9d2c 31 a1                 .wo BBstring-1      ;STRING$
9d2e 15 a1                 .wo BBspaces-1      ;SPACES

9d30 43 e5                 .wo clrscr-1        ;CLS    * Befehle
9d32 63 9a                 .wo setzeichen-1    ;OLDCHAR
9d34 e1 fc                 .wo reset-1         ;C64
9d36 ff 99                 .wo BBxbasic-1      ;XBASIC
9d38 aa 9e                 .wo BBinst-1        ;INST
9d3a 18 9f                 .wo BBdefchar-1     ;DEFCHAR
9d3c 91 9e                 .wo BBold-1         ;OLDCHAR

               RESERVE3:   .db 100+befehle-RESERVE3;Reserve fuer Erweiterung
en
```

Listing 4/6.4.1.5 (Teil 3)

4/6.4.1.6

Die neue LIST-Routine

Um ein Programm auszugeben, wie es bei dem LIST-Befehl geschieht, müssen die Token wieder in Klartext umgewandelt werden. Die Originalroutine erledigt dies natürlich nur für die Standardbefehle des Commodore 64. Wir erweitern daher die Originalroutine mit einer Ausgaberoutine für unsere Befehle.

Bei der Ausgabe durch die LIST-Routine prüfen wir, ob das auszugebende Zeichen das Zeichen mit der Nummer 210 ist. Ist dies nicht der Fall, kann die alte LIST-Routine die Ausgabe übernehmen. Andernfalls handelt es sich um einen unserer Befehle. Die folgende Nummer gibt nun den Tabelleneintrag in der Befehlstabelle an. Leider können wir diesen Eintrag nicht einfach ausgeben, denn der Eintrag kann ja wiederum Token der Originalbefehle enthalten.

Wir geben daher nur die Zeichen mit einem Code, der kleiner als 128 ist, aus . Die anderen Zeichen werden als Token betrachtet, und das zugehörige Befehlswort in der Originaltabelle des Commodore 64 gesucht. Nachdem dieses ausgegeben ist, kann mit unserer Ausgaberoutine fortgefahren werden.

```
                  ;--------------------------------------
                  ;---   Neue LIST Routine             ---
                  ;--------------------------------------
9d84 10 51        list:      bpl normlist1      ;kein Token
9d86 c9 d2                   cmp #xtoken
9d88 d0 50                   bne normlist2      ;kein XBASIC Befehl
9d8a 24 0f                   bit $f             ;Hochkomma ?
9d8c 30 49                   bmi normlist1      ;ja
9d8e c8                      iny                ;XBASIC Nummer holen
9d8f b1 5f                   lda ($5f),y
9d91 aa                      tax                ;Nummer in X bringen
9d92 84 49                   sty $49            ;Y merken
9d94 a0 00                   ldy #0             ;Befehlswort suchen
9d96 ca           suchbef:   dex                ;Nummer-1
9d97 f0 09                   beq foundbef       ;Befehlswort gefunden
9d99 c8           nextbef:   iny                ;naechstes Befehlswort sucher
9d9a b9 20 9c                lda beflist,y
9d9d d0 fa                   bne nextbef
9d9f c8                      iny                ;weitersuchen
9da0 d0 f4                   bne suchbef
9da2 b9 20 9c     foundbef:  lda beflist,y      ;Ausgabe des Befehlswortes
9da5 30 08                   bmi oldnew
9da7 f0 34                   beq normlist3
9da9 20 47 ab                jsr $ab47          ;Ausgabe des Zeichens
```

Listing 4/6.4.1.6-1 (Teil 1)

```
9dac c8                         iny
9dad d0 f3                      bne foundbef
9daf 84 02       oldnew:        sty xmem           ;Ausgabe alter Basic-Befehle
9db1 38                         sec                ;Tokennummer ermitteln
9db2 e9 7f                      sbc #$7f
9db4 aa                         tax
9db5 a0 ff                      ldy #$ff
9db7 ca          wsoldbef:      dex                ;Text suchen
9db8 f0 08                      beq suoldbef
9dba c8          soldbef:       iny
9dbb b9 9e a0                   lda $a09e,y
9dbe 10 fa                      bpl soldbef
9dc0 30 f5                      bmi wsoldbef
9dc2 c8          suoldbef:      iny                ;Befehlswort ausgeben
9dc3 b9 9e a0                   lda $a09e,y
9dc6 30 05                      bmi weiter
9dc8 20 47 ab                   jsr $ab47          ;ein Zeichen ausgeben
9dcb d0 f5                      bne suoldbef
9dcd 29 7f       weiter:        and #$7f           ;letzten Buchstaben
9dcf 20 47 ab                   jsr $ab47          ;ausgeben
9dd2 a4 02                      ldy xmem           ;und weiter bei der neuen ausgabe
9dd4 c8                         iny
9dd5 d0 cb                      bne foundbef
9dd7 4c f3 a6    normlist1:     jmp $a6f3          ;Ruecksprung in die normale L
ist-Routine
9dda 4c 1c a7    normlist2:     jmp $a71c
9ddd a4 49       normlist3:     ldy $49
9ddf 4c f6 a6                   jmp $a6f6
```

Listing 4/6.4.1.6-1 (Teil 2)

Im Anschluß an die LIST-Routine folgt die Einschaltmeldung unserer Basic-Erweiterung. Sie entspricht der Meldung beim Einschalten des Gerätes, jedoch heißt unser Basic 'XBASIC1.0'.

```
                 ;---------------------------------------
                 ;---  XBASIC   Einschaltmeldung      ---
                 ;---------------------------------------
9de2 93 d8 c2    meldung:      .by 147,"XBASIC Version 1.0",13
c1 d3 c9 c3 20 d6 45 52 53 49 4f 4e 20 31 2e 30 0d
9df6 28 43 29                  .by "(c) M.Friese 1987",13,13,0
20 cd 2e c6 52 49 45 53 45 20 31 39 38 37 0d 0d 00
```

Listing 4/6.4.1.6-2

4/6.4.1.7

Die Befehlsausführung

Die Ein- und Ausgabe von Programmzeilen und Direktbefehlen ist damit abgeschlossen. Bei der Ausführung der Befehle holt sich der Basic-Interpreter die Adressen der Befehle aus einer Tabelle. Er benutzt dazu den Aufbau der Programmzeilen. Da jeder Befehl als Token dargestellt ist, und die Nummer des Befehls aus der Tokennummer hervorgeht, können die Befehle so schnell abgearbeitet werden.

Damit unsere Befehle ebenfalls abgearbeitet werden, müssen wir auch in diesen Prozeß eingreifen. Dies muß einmal für die Befehle und außerdem noch einmal für die Funktionen geschehen, da diese an einer anderen Stelle abgearbeitet werden. Im Prinzip funktionieren die beiden Routinen aber gleich.

Befehle werden von der Routine <execute> abgearbeitet, Funktionen hingegen von der Routine <exefkt>. Die Funktionsweise wollen wir anhand der Routine <execute> aufzeigen.

Zunächst wird das nächste Zeichen aus dem Basic-Text mit Hilfe der Routine <chrget> geholt und der Prozessorstatus gerettet. Da unsere Befehle alle mit der Nummer 210 beginnen, brauchen wir das gelesene Zeichen nur mit dieser Nummer vergleichen. Ist unsere Erweiterung nicht gefordert, wird der Prozessorstatus zurückgeholt und bei dem Originalinterpreter weitergemacht.

Andernfalls holen wir das nächste Zeichen, welches die Nummer des Befehls enthält. Dies geschieht ab <xbasictk>. Der normale Basic-Interpreter wird nun abgeschaltet, um auf die Befehle unter dem ROM zugreifen zu können. Damit eine Funktion nicht als Befehl ausgeführt werden kann, vergleichen wir die Nummer des Befehls mit der höchsten Funktionsnummer. Diese einfache Methode ist möglich, da wir gefordert haben, daß alle Funktionsnamen vor den Befehlsnamen in der Tabelle stehen.

Anschließend wird die Befehlsadresse aus der Tabelle geholt und der Befehl aufgerufen. Dieser muß mit 'RTS' enden, damit der Befehl korrekt abgeschlossen wird. Nach dem RTS-Befehl setzt der Prozessor das Programm in Zeile 528 fort (machen Sie sich klar, warum er hier weiterarbeitet). Der normale Basic-Interpreter wird wieder eingeschaltet, und die weitere Abarbeitung an ihn übergeben.

```

                  ;-------------------------------------
                  ;---  XBASIC  Befehlsausfuehrung     ---
                  ;-------------------------------------
9e0a a9 00        exefkt:     lda #0              ;Funktionen aufrufen
9e0c 85 0d                    sta $d
9e0e 20 73 00                 jsr chrget          ;Token holen
9e11 08                       php
9e12 c9 d2                    cmp #xtoken         ;XBASIC-Funktion
9e14 f0 04                    beq exexfkt         ;ja
9e16 28                       plp
9e17 4c 8d ae     falsch:     jmp $ae8d           ;zur normalen Routine

9e1a 28           exexfkt:    plp                 ;Funktionsnummer holen
9e1b 20 73 00                 jsr chrget
9e1e cd 1f 9c                 cmp funktionen      ;Funktion ?
9e21 b0 f4                    bcs falsch          ;nein, ERROR
9e23 c6 01                    dec pp
9e25 20 2b 9e                 jsr execfb          ;Funktion aufrufen
9e28 e6 01                    inc pp
9e2a 60                       rts

9e2b 0a           execfb:     asl                 ;Funktionsnummer mal zwei
9e2c aa                       tax
9e2d bd 1f 9d                 lda befehle-2+1,x   ;Funktionsadresse holen
9e30 48                       pha
9e31 bd 1e 9d                 lda befehle-2,x
9e34 48                       pha
9e35 60                       rts                 ;und Funktion aufrufen

9e36 20 73 00     execute:    jsr chrget          ;Befehl ausfuehren
9e39 08           exeif:      php
9e3a c9 d2                    cmp #xtoken         ;XBASIC-Befehl
9e3c f0 08                    beq xbasictk        ;ja
9e3e c9 8b                    cmp #139            ;IF-Befehl
9e40 f0 25                    beq newif           ;ja, dann zum neuen IF
9e42 28                       plp
9e43 4c e7 a7                 jmp $a7e7           ;sonst zur normalen Routine

9e46 68           xbasictk:   pla                 ;XBASIC-Befehl aufrufen
9e47 c6 01                    dec pp
9e49 20 52 9e                 jsr execxbas
9e4c e6 01                    inc pp
9e4e 4c ae a7                 jmp $a7ae           ;und zur Interpreterschleife

9e51 60           funktion:   rts                 ;Befehl war Funktion

9e52 20 73 00     execxbas:   jsr chrget          ;Befehlsnummer holen
9e55 cd 1f 9c                 cmp funktionen
9e58 90 f7                    bcc funktion        ;Nummer ist eine Funktion
9e5a 0a                       asl                 ;Nummer mal zwei
9e5b aa                       tax                 ;Befehlsadresse holen
9e5c bd 1f 9d                 lda befehle-2+1,x
9e5f 48                       pha
9e60 bd 1e 9d                 lda befehle-2,x
```

Listing 4/6.4.1.7 (Teil 1)

```
9e63 48          pha
9e64 4c 73 00    jmp chrget          ;Befehl ausfuehren
```

Listing 4/6.4.1.7 (Teil 2)

4/6.4.1.8

Probleme mit dem THEN-Befehl

Leider gibt es bei dieser Methode Probleme mit dem THEN-Befehl. Der Basic-Interpreter prüft, ob dem THEN-Befehl eine Zeilennummer oder ein Befehl folgt. Steht hinter dem THEN-Befehl einer unserer Befehle, nimmt der Basic-Interpreter einen Fehler an. Dies kann man vermeiden, wenn man den Befehl durch einen Doppelpunkt von dem THEN-Befehl abtrennt.

Beispiel: Der neue Befehl 'CLS' soll nach einem THEN ausgeführt werden:

```
IF A=1 THEN:CLS
```

Dies ist natürlich nicht besonders elegant. Wir werden daher diesen Fehler beseitigen, indem wir den IF-Befehl durch eine eigene Routine ersetzen. Dazu vergleichen wir die gelesenen Zeichen in der Routine <execute> nicht nur mit 210 (für unsere neuen Befehle), sondern auch mit 139, da dies das Token des IF-Befehls ist.

Unser IF-Befehl ist im wesentlichen eine Kopie des normalen IF-Befehls, da wir ja nur die Ausführung nach dem THEN-Befehl ändern wollen. Soll der Befehl nach dem THEN-Befehl ausgeführt werden, springen wir einfach in unsere Befehlsausführungsroutine. Damit ist der Fehler bereits behoben.

```
               ;--------------------------------------
               ;---  Neuer IF-Befehl                ---
               ;--------------------------------------
9e67 28        newif:     plp                 ;Neuer IF-Befehl
9e68 20 73 00             jsr chrget          ;Kopie der alten Routine
9e6b 20 9e ad             jsr ROMfrmevl
9e6e 20 79 00             jsr chrgot
9e71 c9 89                cmp ##$89           ;GOTO Token
9e73 f0 05                beq isgoto
9e75 a9 a7                lda ##$a7           ;THEN Token
9e77 20 ff ae             jsr testcode
9e7a a5 61     isgoto:    lda $61             ;Ergebnis der IF-Abfrage
9e7c d0 09                bne wahr
9e7e 20 09 a9             jsr $a909           ;zur naechsten Zeile
9e81 20 fb a8             jsr $a8fb
9e84 4c ae a7             jmp $a7ae           ;und weiter im Basic
9e87 20 79 00  wahr:      jsr chrgot
9e8a b0 ad                bcs exeif           ;nach THEN in neue Executeroutine
9e8c 20 a0 a8             jsr $a8a0           ;zum GOTO Befehl
9e8f 4c ae a7             jmp $a7ae           ;und weiter im Basic
```

Listing 4/6.4.1.8-1

Mit dieser Korrektur ist der Systemteil der Basic-Erweiterung abgeschlossen. Es fehlen nur noch die Routinen der Befehle. Diese bestehen aus eigenständigen Unterprogrammen. Um einen Befehl hinzuzufügen, müssen daher nur folgende Schritte durchgeführt werden:

1. Der Befehlsname wird in der Befehlstabelle eingetragen.
2. Die Adresse des Befehls (-1) wird in die Tabelle 'befehle' eingetragen, und zwar an der gleichen Position wie der Befehlsname in der Befehlstabelle, denn aus der Position in der Befehlstabelle gewinnen wir die Nummer des Befehls, und diese Nummer verwenden wir als Index für die Tabelle der Adressen.
3. Die Routine des Befehls wird an das Programm angehängt.

Da das System der Erweiterung im RAM liegen muß, lassen wir den Assembler an dieser Stelle überprüfen, ob wir den reservierten Bereich von 1,5 KByte überschritten haben. Dies ist der Fall, wenn der Programmzähler an dieser Stelle größer als $9FFF ist. Wenn ja, lassen wir den Assembler den Text 'RAM-Bereich überschritten' und die Größe des überschrittenen Bereichs ausgeben.

```
;---------------------------------------
;--- Hier endet der RAM-Bereich. Die ---
;--- folgenden Programmteile liegen  ---
;--- teilweise unter dem Basic-Rom.  ---
;--- Die Zeilen dieses Abschnitts    ---
;--- testen ob der Ram-Bereich bei   ---
;--- der Assemblierung ueberschritten---
;--- wurde.                          ---
;---------------------------------------
           .?p *-$a000        ;Programmzaehler testen
           .be 0              ;Beep
           .tx "RAM-Bereich ueberschritten"
           .oo *-$a000
           .eb

```

Listing 4/6.4.1.8-2

4/6.4.2

Programmierhilfen als Basic-Erweiterung

Wir wollen in der Grundversion einige Programmierhilfen als Basic-Erweiterung zur Verfügung stellen. Die Routinen der Befehle, sowie jeweils ein Beispiel für ihre Anwendung stehen in den nachfolgenden Kapiteln. Hier geben wir zunächst eine Übersicht über die neuen Befehle.

Befehle:

Befehl	Funktion
CLS	löscht den Bildschirm
OLDCHAR	kopiert den Originalzeichensatz in das RAM
C64	schaltet die Basic-Erweiterung ab
XBASIC	startet die Basic-Erweiterung neu
OLD	stellt ein Programm nach NEW wieder her
INST E$, I$,P	überschreibt die Zeichenkette I$ ab Position P mit der Zeichenkette E$
DEFCHAR X,Z,M,A$	Definiert das Zeichen X aus dem Zeichensatz Z neu. A$ enthält die Definition und M den Aufbau der Definition
FRE(X)	wie die Originalfunktion
AT(Z,S)	setze den Cursor auf Zeile Z, Spalte S und liefert "" als Funktionsergebnis
HEX$(X)	liefert X als vierstellige Hexzahl
DEC(X$)	liefert den Dezimalwert der Hexzahl X$
INKEY$	wartet auf einen Tastendruck und liefert die Taste als String. Der Cursor wird dabei eingeschaltet
INSTR(P,S$,IN$)	sucht den String S$ in IN$ ab Position P
STRING$(A$,X)	liefert Xmal den String A$
SPACE$(X)	liefert X Leerzeichen

Für die Befehle CLS, OLDCHAR, XBASIC und C64 existieren keine eigenen Befehlsroutinen. CLS ruft die im Betriebssystem vorhandene Routine zum Löschen des Bildschirms auf. OLDCHAR benutzt die bei der Initialisierung beschriebene Routine zum Kopieren des Zeichensatzes. XBASIC ruft die Initialisierung der Basic-Erweiterung auf und installiert daher auch die Vektoren neu. C64 schließlich ruft die RESET-Routine des Betriebssystems auf. Die Basic-Erweiterung wird dabei ausgekoppelt. Da sie jedoch noch im Speicher vorhanden ist, kann sie mit SYS 39424 wieder gestartet werden. Für die anderen Befehle existieren eigene Routinen.

4/6.4.2.1

OLD

Der NEW-Befehl des Computers zerstört ein Programm im Speicher nicht, sondern trägt am Anfang lediglich zwei Nullen ein, und setzt einige Vektoren in der Zero-Page auf den Speicheranfang. Man kann daher einen versehentlich eingegebenen NEW-Befehl rückgängig machen, solange keine neuen Programmzeilen eingegeben oder Variablen angelegt wurden. Genau dies leistet der OLD-Befehl. Er löscht eine der Nullen am Anfang, und ruft eine Systemroutine zum Binden der Programmzeilen auf. Diese restauriert auch die zweite Null und liefert das Programmende. Dieser Wert wird noch in die Vektoren in der Zero-Page eingesetzt. Anschließend sollte der CLR-Befehl aufgerufen werden.

```
                 ;-------------------------------------
                 ;--- XBASIC-Befehl   OLD            ---
                 ;-------------------------------------
9e92 a5 2c       BBold:     lda 44           ;Kennung am Prg. Anfang loeschen
9e94 a0 01                  ldy #1
9e96 91 2b                  sta (43),y
9e98 a2 12                  ldx #18          ;Programmzeilen neu binden
9e9a 20 b6 9a               jsr basicrom
9e9d a5 22                  lda 34           ;Programmende setzen
9e9f 18                     clc
9ea0 69 02                  adc #2
9ea2 85 2d                  sta 45
9ea4 a5 23                  lda 35
9ea6 69 00                  adc #0
9ea8 85 2e                  sta 46
9eaa 60                     rts
```

Listing 4/6.4.2.1

Beispiel:

10 a=2:b=4:c=a*5+b	Programmzeilen
20 print a,b,c	eingeben
new	Programm löschen
ready.	
list	
ready.	
old	Programm wiederher-
ready.	stellen
clr	
ready.	
list	
10 a=2:b=4:c=a*5+b	
20 print a,b,c	
ready	

4/6.4.2.2

INST A$, B$,P

Dieser Befehl setzt den String A$ in den String B$ ab Position P ein. Dazu werden zunächst die Parameter eingelesen und gespeichert. Als nächstes muß überprüft werden, ob die Aktion möglich ist . Wenn das Ergebnis länger als der Eingabestring B$ ist, kann die Aktion nicht durchgeführt werden und der Befehl bricht mit 'OVERFLOW ERROR' ab. Andernfalls wird der String A$ in den String B$ in der Schleife <putstring> eingesetzt.

```
                        ;--------------------------------------
                        ;--- XBASIC-Befehl  INST A$,B$,P     ---
                        ;--------------------------------------
9eab 20 0b 9b          BBinst:     jsr frmevl        ;Zeichenkette holen
9eae a2 04                         ldx #4
9eb0 20 b6 9a                      jsr basicrom
9eb3 8c 3c 03                      sty mem           ;Laenge in mem
9eb6 a5 22                         lda $22           ;Adresse in azg2
9eb8 85 fd                         sta azg2
9eba a5 23                         lda $23
9ebc 85 fe                         sta azg2+1
9ebe 20 23 9b                      jsr chkkom        ;Stringvariable holen
9ec1 a2 14                         ldx #20
9ec3 20 b6 9a                      jsr basicrom      ;varsuch
9ec6 a5 0d                         lda $d            ;Typflag holen
9ec8 10 47                         bpl typerr
9eca a0 02                         ldy #2            ;azg zeigt auf den String
9ecc b1 47                         lda ($47),y
9ece 85 fc                         sta azg+1
9ed0 88                            dey
9ed1 b1 47                         lda ($47),y
9ed3 85 fb                         sta azg
9ed5 20 23 9b                      jsr chkkom        ;Position einlesen
9ed8 20 33 9b                      jsr getbyte
9edb ca                            dex               ;Index 1.. -> 0..
9edc e0 ff                         cpx #$ff
9ede d0 05                         bne *+2+2+3
9ee0 a2 0e                         ldx #$e           ;falscher Index (0)
9ee2 4c 06 9b                      jmp fehler
9ee5 8e 3d 03                      stx mem+1         ;Position merken
9ee8 18                            clc               ;minimale Stringlaenge berechnen
9ee9 8a                            txa               ;Position
9eea 6d 3c 03                      adc mem           ;+Laenge der Zeichenkette
9eed b0 25                         bcs strerr
9eef e9 00                         sbc #0            ;-1 ergibt minimale Laenge fuer B$
```

Listing 4/6.4.2.2 (Teil 1)

```
9ef1 a0 00                 ldy #0            ;mit tatsaechlicher Laenge
9ef3 d1 47                 cmp ($47),y       ;vergleichen
9ef5 b0 1d                 bcs strerr        ;zu kurz
9ef7 ad 3d 03              lda mem+1         ;Zeiger auf Einfuegeposition
9efa 65 fb                 adc azg           ;berechnen
9efc 85 fb                 sta azg
9efe 90 02                 bcc *+2+2
9f00 e6 fc                 inc azg+1
9f02 ac 3c 03              ldy mem           ;Zeichenkette einsetzen
9f05 ae 3c 03              ldx mem
9f08 88        putinstring:dey
9f09 b1 fd                 lda (azg2),y
9f0b 91 fb                 sta (azg),y
9f0d ca                    dex               ;bis alle Zeichen eingesetzt
9f0e d0 f8                 bne putinstring
9f10 60                    rts

9f11 a2 16     typerr:     ldx #22
9f13 2c                    bit
9f14 a2 0f     strerr:     ldx #15
9f16 4c 06 9b              jmp fehler

```

Listing 4/6.4.2.2 (Teil 2)

Beispiel:

a$="Dies istabcdeTest"
ready.
inst" ein ",a$,9
ready.
?a$
Dies ist ein Test
ready.

4/6.4.2.3

DEFCHAR NR, ZS, MO, Z$

DEFCHAR definiert ein beliebiges Zeichen um. Dabei bezeichnet NR die Nummer des Zeichens innerhalb des Zeichensatzes. Diese Nummer ist aus der Tabelle 'Bildschirmn-Codes' des C 64-Handbuches zu entnehmen. ZS bezeichnet den Zeichensatz. Dabei steht eine Null für den Großschrift/Graphik-Zeichensatz und eine Eins für den Klein-/Großschrift-Zeichensatz. Z$ ist eine Zeichenkette, die für jeden Punkt die Information enthält, ob er gesetzt werden soll. Dabei bestimmt MO den Aufbau von Z$. MO legt also den Arbeitsmodus fest.

Ist MO=0, so muß Z$ aus 64 Zeichen bestehen. Jedes Zeichen steht dann für einen der 64 Punkte aus der 8*8 Punkte großen Zeichendefinition. Soll ein Punkt gesetzt werden, so ist ein 'A' anzugeben. Ein gelöschter Punkt wird durch das Zeichen '.' repräsentiert.

Ist MO=1, besteht Z$ lediglich aus 32 Zeichen. Jedes Zeichen ist dann für zwei nebeneinanderliegende Punkte zuständig. Dabei wurde folgende Zuordnung getroffen:

Punkte	Zeichen
00	.
01	A
10	B
11	C

In diesem Modus lassen sich leicht Zeichen im Multicolormodus erzeugen. Jedes Zeichen entspricht dann einer Farbe.

Der Modus 2 (bzw. 3) entspricht dem Modus 0 (bzw. 1), jedoch wird hier das erzeugte Zeichen invertiert. Sie können so mit einer Zeichendefinition auch das inverse Zeichen erzeugen.

Die Funktionsweise des Befehls ist recht einfach: Zunächst wird die Adresse der Zeichendefinition errechnet, indem zu der Basisadresse des Zeichengenerators die Nummer des Zeichens mal 8 addiert wird, da ein Zeichen aus 8 Byte besteht. Ist der Zeichensatz 1 gewünscht, muß noch 2048 addiert werden, da die Gesamtlänge eines Zeichensatzes 256 Zeichen * 8 Byte = 2048 Byte beträgt. Die Adresse der Zeichendefinition errechnet sich also nach der Formel:

$D000 + NR * 8 + ZS * 2048.

Ab dieser Adresse werden nun 8 Byte abgelegt, wobei sich jedes Byte aus dem zugehörigen Zeichen im String, und dem Modus ergibt. Steht ein falsches Zeichen im String, wird die Meldung 'bad char in string' ausgegeben. Am Ende des Befehls wird die Stringlänge mit den gelesenen, d.h. für die Definition benötigten Zeichen, verglichen. Weichen sie voneinander ab, wird der Fehler 'wrong stringlen' gemeldet. Das Zeichen ist dann bereits umdefiniert.

Der Befehl OLDCHAR macht alle Änderungen wieder rückgängig.

```
                ;-------------------------------------
                ;--- XBASIC-Befehl DEFCHAR X,Y,A$   ---
                ;-------------------------------------
9f19 20 33 9b   BBdefchar: jsr getbyte      ;Zeichenummer holen
9f1c 86 fb                 stx azg
9f1e a9 00                 lda #0
9f20 85 fc                 sta azg+1        ;mal 8
9f22 06 fb                 asl azg
9f24 26 fc                 rol azg+1
```

Listing 4/6.4.2.3 (Teil 1)

```
9f26 06 fb                asl azg
9f28 26 fc                rol azg+1
9f2a 06 fb                asl azg
9f2c 26 fc                rol azg+1
9f2e 18                   clc                ;+Basisadresse
9f2f a9 00                lda #<zeichensatz
9f31 65 fb                adc azg
9f33 85 fb                sta azg
9f35 a9 d0                lda #>zeichensatz  ;ergibt Zeichenadresse in azg
9f37 65 fc                adc azg+1
9f39 85 fc                sta azg+1
9f3b a9 08                lda #8             ;8 Bytes pro Zeichen
9f3d 8d 3e 03             sta anzbytes
9f40 20 23 9b             jsr chkkom         ;Zeichensatznummer holen
9f43 20 33 9b             jsr getbyte
9f46 e0 02                cpx #2
9f48 b0 12                bcs zugrol
9f4a 8a                   txa                ;Adresse+2048*Zeichensatz
9f4b 0a                   asl
9f4c 0a                   asl
9f4d 0a                   asl
9f4e 65 fc                adc azg+1
9f50 85 fc                sta azg+1
9f52 20 23 9b    define:  jsr chkkom         ;Flag fuer Modus holen
9f55 20 33 9b             jsr getbyte
9f58 e0 04                cpx #4
9f5a 90 03                bcc *+3+2
9f5c 4c 11 a1    zugrol:  jmp zugross
9f5f 8a                   txa
9f60 29 01                and #1
9f62 85 fd                sta azg2           ;Flag fuer Doppelt-Modus
9f64 8a                   txa
9f65 4a                   lsr
9f66 85 fe                sta azg2+1         ;Flag fuer Invers
9f68 a9 00                lda #0             ;Bytezaehler auf Null
9f6a 8d 3d 03             sta posi
9f6d 20 23 9b             jsr chkkom
9f70 20 0b 9b             jsr frmevl         ;String holen
9f73 a2 04                ldx #4
9f75 20 b6 9a             jsr basicrom
9f78 8d 3c 03             sta strlen         ;Stringlaenge merken
9f7b a0 ff                ldy #$ff           ;Y zeigt auf String
9f7d a2 08       bytes:   ldx #8             ;8 Bits pro Byte
9f7f 86 02                stx xmem
9f81 a6 fd                ldx azg2           ;Doppelt-Modus?
9f83 f0 02                beq by
9f85 46 02                lsr xmem           ;ja, dann nur 4 Zeichen pro Byte
9f87 c8          by:      iny
9f88 b1 22                lda ($22),y
9f8a c9 2e                cmp #'.            ;Zeichen ueberpruefen
9f8c f0 14                beq null
9f8e 38                   sec
9f8f e9 40                sbc #'a-1
9f91 90 5d                bcc badchar
9f93 f0 5b                beq badchar
9f95 c9 04                cmp #4
9f97 b0 57                bcs badchar
```

Listing 4/6.4.2.3 (Teil 2)

```
9f99 a6 fd                    ldx azg2
9f9b d0 07                    bne doppel1
9f9d c9 02                    cmp #2
9f9f b0 4f                    bcs badchar
9fa1 2c                       bit
9fa2 a9 00        null:       lda #0
9fa4 a6 fd        doppel1:    ldx azg2
9fa6 d0 07                    bne doppelt3
9fa8 6a                       ror               ;ein Bit uebernehmen
9fa9 2e 3f 03                 rol byte
9fac 4c b8 9f                 jmp makeabyte
9faf 6a           doppelt3:   ror               ;zwei Bit uebernehmen
9fb0 6a                       ror
9fb1 2e 3f 03                 rol byte
9fb4 2a                       rol
9fb5 2e 3f 03                 rol byte
9fb8 c6 02        makeabyte:  dec xmem          ;weiter bis ein Byte fertig
9fba d0 cb                    bne by
9fbc 98                       tya               ;Y merken
9fbd 48                       pha
9fbe a5 01                    lda pp            ;Zeichenbereich einschalten
9fc0 48                       pha               ;(alles RAM)
9fc1 78                       sei
9fc2 a9 32                    lda #$32
9fc4 85 01                    sta pp
9fc6 ac 3d 03                 ldy posi          ;Bytenummer holen
9fc9 ad 3f 03                 lda byte          ;Byte speichern
9fcc a6 fe                    ldx azg2+1        ;Invers
9fce f0 02                    beq putb          ;nein
9fd0 49 ff                    eor #$ff
9fd2 91 fb        putb:       sta (azg),y
9fd4 68                       pla               ;Speicherbereich normal
9fd5 85 01                    sta pp
9fd7 58                       cli
9fd8 ee 3d 03                 inc posi          ;Bytenummer+1
9fdb 68                       pla               ;Y holen
9fdc a8                       tay
9fdd ce 3e 03                 dec anzbytes      ;bis alle Bytes gesetzt
9fe0 d0 9b                    bne bytes
9fe2 c8                       iny
9fe3 cc 3c 03                 cpy strlen        ;String vollstaendig abgearbeitet
9fe6 d0 01                    bne strlenerr     ;nein
9fe8 60                       rts

9fe9 a2 e5        strlenerr:  ldx #<slerrtxt    ;falsche Stringlaenge
9feb a9 9a                    lda #>slerrtxt
9fed 4c de 9a                 jmp adrfehler

9ff0 a2 f4        badchar:    ldx #<bcerrtxt    ;falsches Zeichen im String
9ff2 a9 9a                    lda #>bcerrtxt
9ff4 4c de 9a                 jmp adrfehler

                  strlen:     .eq mem           ;Stringlaenge
                  posi:       .eq mem+1         ;Byteposition
                  anzbytes:   .eq mem+2         ;Anzahl Bytes der Zeichen
                  byte:       .eq mem+3         ;Speicher fuer ein Byte
```

Listing 4/6.4.2.3 (Teil 3)

Als Beispiel soll das Zeichen '<' in das Zeichen '©' gewandelt werden:

```
1     A$=”..AAAA..”
2 A$=A$+”.A....A.”
3 A$=A$+”A..AAA.A”
4 A$=A$+”A.A....A”
5 A$=A$+”A.A....A”
6 A$=A$+”A..AAA.A”
7 A$=A$+”.A....A.”
8 A$=A$+”..AAAA..”
9 DEFCHAR 60,0,0,A$:PRINT”<”
RUN
©
READY.
```

4/6.4.2.4

A=FRE(X)

Die FRE-Funktion kennen Sie bereits vom Standardbasic des C 64. Aber Sie wissen sicherlich auch, daß diese Funktion nur bei Werten bis zu 32768 richtig funktioniert. Dies liegt daran, daß Commodore die Ausgaberoutine für Integerwerte benutzt hat, welche größere Werte als negative Zahlen interpretiert. Dies wollen wir hier korrigieren, indem wir die Routine kopieren und mit einer anderen Ausgaberoutine abschließen. Um unsere Forderung erfüllen zu können, daß ein Befehl aus mindestens zwei Zeichen besteht, ist der Befehl unter dem Namen 'FREC' in der Befehlstabelle eingetragen worden, da die Zeichenkette 'FRE' ja als ein Zeichen (ein Token) dargestellt wird. Daher ruft 'FRE(0)' die neue Routine, und 'FRE (0)' die alte Routine auf.

```
                ;--------------------------------------
                ;--- XBASIC-Befehl   A=FRE(A)       ---
                ;--------------------------------------
9ff7 20 73 00   BBfre:   jsr chrget
9ffa 20 0b 9b            jsr frmevl        ;Wert holen
9ffd 20 13 9b            jsr chkklazu      ;Klammer zu pruefen
a000 a9 0d               lda #$d           ;Typ testen
a002 f0 05               beq nstring
```

Listing 4/6.4.2.4 (Teil 1)

```
a004 a2 10                  ldx #16
a006 20 b6 9a               jsr basicrom        ;frestr
a009 a2 02      nstring:    ldx #2              ;garbage collection
a00b 20 b6 9a               jsr basicrom
a00e 38                     sec                 ;Freien Bereich berechnen
a00f a5 33                  lda $33
a011 e5 31                  sbc $31
a013 a8                     tay
a014 a5 34                  lda $34
a016 e5 32                  sbc $32
a018 a2 00      ergebnis:   ldx #0              ;Typ numerisch
a01a 86 0d                  stx $d
a01c 85 62      outpos:     sta $62             ;Ergebnis in FAC bringen
a01e 84 63                  sty $63
a020 a2 90                  ldx #$90
a022 86 02                  stx xmem
a024 38                     sec
a025 a2 08                  ldx #8
a027 4c b6 9a               jmp basicrom
```

Listing 4/6.4.2.4 (Teil 2)

Beispiel:

```
NEW
?FRE (0)
-28163
READY.
?FRE (0)
37373
READY.
```

4/6.4.2.5

Zahlenumwandlung

Häufig benötigt man auch in Basic-Programmen hexadezimale Zahlen. Die Umwandlung in Basic dauert im allgemeinen recht lange. Daher wollen wir uns zwei Befehle zur Verfügung stellen, welche die Umwandlung zwischen dezimalen und hexadezimalen Zahlen ermöglichen.

A=DEC(A$)

Als erstes wollen wir eine maximal vierstellige Zeichenkette, welche eine hexadezimale Zahl darstellt, in eine Dezimalzahl umrechnen. Ist die Zeichenkette länger als vier Zeichen, sollen nur die vier letzten Zeichen beachtet werden. Die Umwandlung erfolgt nach dem folgenden Algorithmus:

1. Das Ergebnis ist Null.
2. Lese ein Zeichen.
3. Wandle '0' bis '9' nach 0 bis 9 und 'A' bis 'F' nach 10 bis 15.
4. Multipliziere das bisherige Ergebnis mit 16 und addiere das Ergebnis von Schritt 3.
5. Weiter beim 2. Schritt bis der String beendet ist.

Als Beispiel soll die Hexadezimalzahl 'AF45' in eine dezimale Zahl gewandelt werden:

gelesenes Zeichen	entspricht der Zahl	Ergebnis
A	10	10
F	15	175
4	4	2804
5	5	44869

Beispiel:

```
?DEC("AF45")
44869
ready.
```

```
                  ;-------------------------------------
                  ;--- XBASIC-Befehl    A=DEC(A$)     ---
                  ;-------------------------------------
a02a 20 73 00     BBdez:     jsr chrget
a02d 20 5e 9b                jsr klammer          ;Klammerausdruck holen
a030 a2 04                   ldx #4               ;String holen
a032 20 b6 9a                jsr basicrom
a035 d0 05                   bne *+2+2+3          ;String =""
a037 a2 06                   ldx #6               ;ja, Ergebnis Null
a039 4c b6 9a                jmp basicrom
a03c 98                      tya                  ;Laenge in X
a03d aa                      tax
a03e a0 00                   ldy #0               ;Ergebnis=0
a040 84 14                   sty $14
a042 84 15                   sty $15
a044 b1 22        dezi:      lda ($22),y          ;Zeichen aus String holen
a046 c9 30                   cmp #'0
a048 90 20                   bcc dezerr           ;kleiner '0 -> ERROR
a04a e9 30                   sbc #'0              ;nach 0.. wandeln
a04c c9 0a                   cmp #9+1
```

Listing 4/6.4.2.5-1 (Teil 1)

```
a04e 90 08              bcc okdez      ;kleiner 10 dann 0..9
a050 e9 07              sbc #7         ;'a..'f wandeln
a052 90 16              bcc dezerr     ;kleiner 'a ->ERROR
a054 c9 10              cmp #16        ;groesser 'f -> ERROR
a056 b0 12              bcs dezerr
a058 20 71 a0  okdez:   jsr asl4       ;Ergebnis mal 16
a05b 18                 clc            ;+ gelesener Wert 0..15
a05c 65 14              adc $14
a05e 85 14              sta $14
a060 a5 15              lda $15
a062 69 00              adc #0
a064 85 15              sta $15
a066 c8                 iny            ;Pos. im String +1
a067 ca                 dex            ;Stelle -1
a068 d0 da              bne dezi       ;weiter
a06a a5 15     dezerr:  lda $15        ;Fertig oder Error
a06c a4 14              ldy $14        ;Ergebnis in FAC liefern
a06e 4c 1c a0           jmp outpos
a071 06 14     asl4:    asl $14        ;Integer in $14/$15
a073 26 15              rol $15        ;4mal linksschieben
a075 06 14              asl $14
a077 26 15              rol $15
a079 06 14              asl $14
a07b 26 15              rol $15
a07d 06 14              asl $14
a07f 26 15              rol $15
a081 60                 rts
```

Listing 4/6.4.2.5-1 (Teil 2)

A$=HEX$(A)

Die Funktion HEX$ ist die Umkehrung von DEC. Er liefert eine Zahl als hexadezimale Zeichenkette. Erreicht wird dies, indem die Zahl zunächst als 16-Bit Binärzahl dargestellt wird. Dann werden die untersten 4 Bit in die Zeichen '0' bis 'F' gewandelt und in den String geschrieben. Nun wird die Zahl um 4 Stellen nach rechts verschoben. Nach viermaligem Ausführen dieser Schritte ist die Zeichenkette fertig erstellt.

```
               ;---------------------------------------
               ;--- XBASIC-Befehl    A$=HEX$(A)      ---
               ;---------------------------------------
a082 20 73 00  BBhex:   jsr chrget
a085 20 5e 9b           jsr klammer    ;Klammerausdruck holen
a088 20 2b 9b           jsr adresse    ;in pos. Integer wandeln
a08b a9 04              lda #4         ;String der Laenge 4 anlegen
a08d 20 4e 9b           jsr reservstr
a090 a0 03              ldy #3         ;vier Stellen
```

Listing 4/6.4.2.5-2 (Teil 1)

```
a092 a5 14      tohex:   lda $14          ;Integerzahl
a094 29 0f               and #%1111       ;Bit 0..3 auswerten
a096 18                  clc              ;nach 0..F wandeln
a097 69 30               adc #'0
a099 c9 3a               cmp #'9+1
a09b 90 02               bcc ziffer
a09d 69 06               adc #6
a09f 91 62      ziffer:  sta ($62),y      ;und in String speichern
a0a1 20 aa a0            jsr asr4         ;Integerzahl/16
a0a4 88                  dey              ;Stellen-1
a0a5 10 eb               bpl tohex        ;bis 4 Stellen fertig
a0a7 4c 56 9b            jmp string       ;String in Stringstack

a0aa 46 15      asr4:    lsr $15          ;Integerzahl in $14/$15
a0ac 66 14               ror $14          ;vier mal rechtsschieben
a0ae 46 15               lsr $15
a0b0 66 14               ror $14
a0b2 46 15               lsr $15
a0b4 66 14               ror $14
a0b6 46 15               lsr $15
a0b8 66 14               ror $14
a0ba 60                  rts
```

Listing 4/6.4.2.5-2 (Teil 2)

Beispiel:

?HEX$(44869)
AF45
READY.

4/6.4.2.6

A$=INKEY$

Basic-Programmierer schreiben sich häufig eigene Eingaberoutinen, die mit dem GET-Befehl arbeiten. Diese Methode hat den Nachteil, daß der Cursor nicht sichtbar ist. Außerdem muß das Warten auf einen Tastendruck extra programmiert werden. Um hier eine Hilfestellung zu haben, wurde die Funktion INKEY$ entwickelt.

Die Routine schaltet zunächst den Cursor an und ruft dann die GET-Funktion auf. Liefert sie den Leerstring, wird GET erneut aufgerufen, bis der Anwender eine Taste drückt. Dann wird der Cursor wieder abgeschaltet und das eingegebene Zeichen als Ergebnis geliefert.

```
                ;--------------------------------------
                ;--- XBASIC-Befehl   A$=INKEY$       ---
                ;--------------------------------------
a0bb a9 00      BBinkey:   lda #0          ;Cursor anschalten
a0bd 85 cf                 sta 207
a0bf 85 cc                 sta 204
a0c1 a2 0c      inkey:     ldx #12         ;Get-Funktione
a0c3 20 b6 9a              jsr basicrom
a0c6 c9 00                 cmp #0          ;nichts gelesen?
a0c8 f0 f7                 beq inkey       ;dann warten
a0ca 48                    pha             ;Zeichen merken
a0cb a9 01                 lda #1          ;Cursor aus
a0cd 85 cc                 sta 204
a0cf a5 ce                 lda 206
a0d1 a4 d3                 ldy 211         ;Zeichen unter Cursor
a0d3 91 d1                 sta (209),y     ;neu setzen
a0d5 a9 01                 lda #1          ;String anlegen
a0d7 20 4e 9b              jsr reservstr
a0da a0 00                 ldy #0          ;Zeichen einsetzen
a0dc 68                    pla
a0dd 91 62                 sta ($62),y
a0df 20 56 9b              jsr string      ;String ist Ergebnis
a0e2 4c 73 00              jmp chrget
```

Listing 4/6.4.2.6

Beispiel:

?INKEY$ (TASTE 'A' drücken)
A
READY.

4/6.4.2.7

A$=AT(X,Y)

Ebenfalls häufig benötigt ist das Positionieren des Cursors auf dem Bildschirm. Dies läßt das Standardbasic leider nicht zu. Man kann diese Funktion aber mit POKE und SYS erreichen. Wir wollen hier einen Befehl 'AT' schreiben, der den Cursor auf dem Bildschirm beliebig positioniert. Außerdem soll es möglich sein, den Befehl in einen PRINT-Befehl einzusetzen.

Beispiel:
PRINT AT(10,20) "TEST"; AT(1,1) "AT-DEMO";

Dies ist jedoch nur möglich, wenn der Befehl AT eine Funktion ist. Wir erstellen AT daher als Funktion, die den Cursor setzt, und als Ergebnis einen Leerstring liefert.

```
                  ;--------------------------------------
                  ;--- XBASIC-Befehl   A$=AT(X,Y)      ---
                  ;--------------------------------------
a0e5 20 73 00     BBat:     jsr chrget       ;Zeile einlesen
a0e8 20 33 9b               jsr getbyte
a0eb e0 19                  cpx #25          ;ueberpruefen
a0ed b0 22                  bcs zugross
a0ef 86 fb                  stx azg          ;und merken
a0f1 20 23 9b               jsr chkkom
a0f4 20 33 9b               jsr getbyte      ;Spalte einlesen
a0f7 e0 28                  cpx #40          ;ueberpruefen
a0f9 b0 16                  bcs zugross
a0fb 86 d3                  stx 211          ;Position setzen
a0fd a6 fb                  ldx azg
a0ff 86 d6                  stx 214
a101 a2 0e                  ldx #14          ;Cursor auf Pos. setzen
a103 20 b6 9a               jsr basicrom
a106 20 13 9b               jsr chkklazu     ;Auf ')' testen
a109 a9 00                  lda #0           ;Ergebnis der Funktion ist ""
a10b 20 4e 9b               jsr reservstr
a10e 4c 56 9b               jmp string

a111 a2 0e        zugross:  ldx #$e
a113 4c 06 9b               jmp fehler

```

Listing 4/6.4.2.7

4/6.4.2.8

A$=SPACE$(X)

Nun wollen wir uns noch ein paar Funktionen definieren, welche den Umgang mit Strings erleichtern. Der Befehl SPACE$ soll uns eine Anzahl von Leerzeichen liefern. Dazu reserviert er zunächst einen String, der durch X angegebenen Länge. Dieser String wird dann in der Schleife <putspaces> mit Leerzeichen gefüllt. Zum Schluß wird der Zeiger an den Basic-Interpreter übergeben.

```
              ;--------------------------------------
              ;--- XBASIC-Befehl   A$=SPACE$(X)    ---
              ;--------------------------------------
a116 20 73 00 BBspaces:  jsr chrget
a119 20 33 9b            jsr getbyte       ;Anzahl holen
a11c 8a                  txa               ;Anzahl merken
a11d 48                  pha               ;String anlegen
a11e 20 4e 9b            jsr reservstr
a121 68                  pla
a122 a8                  tay               ;Leerzeichen in String schreiben
a123 f0 07               beq leerstr
a125 a9 20               lda #32
a127 88       putspaces: dey
a128 91 62               sta ($62),y
a12a d0 fb               bne putspaces     ;String als Ergebnis liefern
a12c 20 56 9b leerstr:   jsr string
a12f 4c 13 9b            jmp chkklazu
```

Listing 4/6.4.2.8

Beispiel:

```
?SPACE$(8);"TEST"
        TEST
READY.
```

4/6.4.2.9

A$=STRING$(A$,X)

Dieser Befehl ist eine Erweiterung von SPACE$. Er liefert x-mal den String A$. Dazu wird ein String der Länge X*LEN(A$) reserviert. Ist die Länge Null, wird der Leerstring zurückgeliefert. Bei der Berechnung wird der Einfachheit halber die Multiplikation auf mehrere Additionen zurückgeführt. Dies ist nicht langsamer als eine Multiplikation, da der String A$ in der Praxis selten länger als ca. 10 Zeichen ist.

In den reservierten String wird dann x-mal der String A$ eingetragen. Dies geschieht ab <makestring>.

```
                    ;---------------------------------------
                    ;--- XBASIC-Befehl  A$=STRING$(A$,X) ---
                    ;---------------------------------------
a132 20 73 00       BBstring:    jsr chrget
a135 20 0b 9b                    jsr frmevl        ;String holen
a138 a2 04                       ldx #4
a13a 20 b6 9a                    jsr basicrom
a13d a5 22                       lda $22
a13f 85 fd                       sta azg2
a141 a5 23                       lda $23
a143 85 fe                       sta azg2+1
a145 84 fb                       sty azg           ;Stringlaenge merken
a147 20 23 9b                    jsr chkkom        ;auf Komma pruefen
a14a 20 33 9b                    jsr getbyte       ;Anzahl der String holen
a14d 86 fc                       stx azg+1         ;Anzahl der Strings merken
a14f 8a                          txa               ;Anzahl der String=0
a150 f0 35                       beq leerstring    ;fuehrt zu einen Leerstring
a152 a6 fb                       ldx azg           ;Stringlaenge
a154 f0 31                       beq leerstring
a156 a9 00                       lda #0
a158 18                          clc
a159 65 fc          multiply:    adc azg+1         ;mit Anzahl Strings
a15b b0 37                       bcs strzugross    ;multiplizieren
a15d ca                          dex
a15e d0 f9                       bne multiply
a160 20 4e 9b                    jsr reservstr     ;Gesamtstring reservieren
a163 a6 fc                       ldx azg+1         ;Anzahl Strings in X
a165 a0 00                       ldy #0            ;Pointer auf Strings=0
a167 84 fc                       sty azg+1
a169 b1 fd          makestrings:lda (azg2),y       ;String ubertragen
a16b 84 02                       sty xmem
a16d a4 fc                       ldy azg+1         ;Pointer auf Gesamtstring
a16f 91 62                       sta ($62),y
```

Listing 4/6.4.2.9 (Teil 1)

```
a171 a4 02     984              ldy xmem
a173 e6 fc     985              inc azg+1
a175 c8        986              iny
a176 c4 fb     987              cpy azg             ;String einmal kopiert ?
a178 d0 ef     988              bne makestrings     ;nein
a17a a0 00     989              ldy #0              ;Noch einmal kopieren?
a17c ca        990              dex
a17d d0 ea     991              bne makestrings     ;ja
a17f c6 0d     992              dec $d
a181 20 56 9b  993              jsr string
a184 4c 13 9b  994              jmp chkklazu
               995
a187 a9 00     996  leerstring:lda #0               ;Leerstring liefern
a189 20 4e 9b  997              jsr reservstr
a18c c6 0d     998              dec $d
a18e 20 56 9b  999              jsr string
a191 4c 13 9b  1000             jmp chkklazu
               1001
a194 a2 17     1002 strzugross:ldx #23              ;String zu lang
a196 4c 06 9b  1003             jmp fehler
               1004
```

Listing 4/6.4.2.9 (Teil 2)

Beispiel:

?STRING$("MANFRED",4)
MANFREDMANFREDMANFREDMANFRED
READY.

4/6.4.2.10

A=INSTR(P,A$,B$)

Als letzten Befehl der Grundversion betrachten wir den INSTR-Befehl. Er sucht den String A$ im String B$. Dies geschieht durch einfaches, zeichenweises Vergleichen des Strings A$ mit einem entsprechend langen Teilstring von B$. Ist der String gefunden, wird die Position des Strings als Wert geliefert. Andernfalls liefert die Funktion den Wert Null. Der Parameter P gibt an, ab welcher Position in B$ mit der Suche begonnen wird.

```
                    ;-----------------------------------------
                    ;--- XBASIC-Befehl  A=INSTR(P,A$,B$) ---
                    ;-----------------------------------------
a199 20 73 00       BBinstr:    jsr chrget
a19c 20 33 9b                   jsr getbyte      ;Position holen
a19f 8e 3c 03                   stx mem          ;und merken
a1a2 20 23 9b                   jsr chkkom
a1a5 20 0b 9b                   jsr frmevl       ;String holen
a1a8 a2 04                      ldx #4
a1aa 20 b6 9a                   jsr basicrom
a1ad a5 22                      lda $22          ;Adresse in azg
a1af 85 fb                      sta azg
a1b1 a5 23                      lda $23
a1b3 85 fc                      sta azg+1
a1b5 8c 3d 03                   sty mem+1        ;Stringlaenge merken
a1b8 20 23 9b                   jsr chkkom
a1bb 20 0b 9b                   jsr frmevl       ;String2 holen
a1be a2 04                      ldx #4
a1c0 20 b6 9a                   jsr basicrom
a1c3 a5 22                      lda $22          ;Adresse in azg2
a1c5 85 fd                      sta azg2
a1c7 a5 23                      lda $23
a1c9 85 fe                      sta azg2+1
a1cb 8c 3e 03                   sty mem+2        ;Stringlaenge merken
a1ce 20 13 9b                   jsr chkklazu
a1d1 ac 3c 03                   ldy mem
a1d4 d0 05                      bne *+2+2+3
a1d6 a2 0e                      ldx #$e          ;Index=0->Fehler
a1d8 4c 06 9b                   jmp fehler
a1db 88                         dey              ;Index 1.. -> 0..
a1dc 8c 41 03                   sty mem+5        ;Aktuelle Suchposition
a1df a9 00          dg:         lda #0           ;Vergleichsposition
a1e1 8d 40 03                   sta mem+4
a1e4 ad 41 03                   lda mem+5        ;Suchen ab Pos. mem+5
a1e7 8d 3f 03                   sta mem+3
a1ea ac 3f 03       strvergl:   ldy mem+3        ;Stringlaenge ueberschritten?
a1ed cc 3e 03                   cpy mem+2
a1f0 b0 1b                      bcs notfound?    ;ja
a1f2 b1 fd                      lda (azg2),y     ;Zeichen holen
a1f4 ac 40 03                   ldy mem+4        ;Laenge des Suchworts uberschritten ?
a1f7 cc 3d 03                   cpy mem+1
a1fa f0 1f                      beq found        ;ja, dann Wort gefunden
a1fc d1 fb                      cmp (azg),y      ;Zeichen vergleichen
a1fe d0 08                      bne durchgang    ;ungleich -> neue Suchposition
a200 ee 3f 03                   inc mem+3        ;naechten Buchstaben vergleichen
a203 ee 40 03                   inc mem+4
a206 d0 e2                      bne strvergl     ;weiter vergleichen

a208 ee 41 03       durchgang:  inc mem+5        ;neue Suchposition
a20b d0 d2                      bne dg           ;und neu vergleichen

a20d ac 40 03       notfound?:  ldy mem+4        ;Wortende erreicht?
a210 cc 3d 03                   cpy mem+1
a213 f0 06                      beq found        ;ja, dann Wort gefunden
a215 a9 00                      lda #0           ;sonst nicht gefunden -> 0 liefern
a217 a8                         tay
a218 4c 18 a0                   jmp ergebnis

```

Listing 4/6.4.2.10 (Teil 1)

```
a21b a9 00     1063  found:    lda #0           ;Position mem+5 liefern
a21d ac 41 03 1064             ldy mem+5
a220 c8        1065            iny
a221 4c 18 a0 1066             jmp ergebnis
               1067
```

Listing 4/6.4.2.10 (Teil 2)

Beispiel:

```
A$="TESTESTABCDEF"
?INSTR(1,"TEST",A$)
1
READY.
?INSTR(2,"TEST",A$)
4
READY.
?INSTR(5,"TEST",A$)
0
READY.
```

Basic-Erweiterung, Version 1.1

Mit der Version 1.1 unserer Basic-Erweiterung XBASIC wird der Befehlssatz um 23 Befehle auf insgesamt 38 Befehle erweitert. Eine Übersicht aller Befehle finden Sie in Kapitel 4/6.4.9.

Bevor wir uns mit den einzelnen Befehlsroutinen auseinandersetzen können, müssen wir unsere Systemroutinen um einige neue Möglichkeiten erweitern. Dazu stellen wir uns weitere ROM-Routinen zur Verfügung, welche im wesentlichen den Datentransfer mit externen Geräten, wie z.B. der Diskettenstation, übernehmen.

Die neuen Routinen werden einfach an die Tabelle ab <rombefehle> angehängt. Wenn Sie diese Änderungen in ihrem Quelltext vornehmen, beachten Sie bitte, daß die Zeilennummern hier nicht mehr übereinstimmen, da für die Ausgabe einige Steuerbefehle eingefügt wurden. Richten Sie sich nur nach den Kapitelüberschriften bzw. Symbolen, oder nach den Adressen der übersetzten Version.

```
                rombefehle:
9b67 fe ae                 .wo testcode-1     ;0
9b69 25 b5                 .wo garbage-1      ;2
9b6b 81 b7                 .wo getstring-1    ;4
9b6d f6 b8                 .wo $b8f7-1        ;6
9b6f 48 bc                 .wo $bc49-1        ;8
9b71 3a ab                 .wo CURrechts-1    ;10
9b73 e3 ff                 .wo get-1          ;12
9b75 6b e5                 .wo $e56c-1        ;14
9b77 a5 b6                 .wo frestr-1       ;16
9b79 32 a5                 .wo 42291-1        ;18
9b7b 8a b0                 .wo varsuch-1      ;20
9b7d dc bd                 .wo $bddd-1        ;22
9b7f 7b a5                 .wo $a57c-1        ;24
9b81 b9 ff                 .wo $ffba-1        ;26 Fileparameter
9b83 bc ff                 .wo $ffbd-1        ;28 Filename
9b85 bf ff                 .wo $ffc0-1        ;30 Open
9b87 c2 ff                 .wo $ffc3-1        ;32 Close
9b89 c5 ff                 .wo $ffc6-1        ;34 Chkin
9b8b c8 ff                 .wo $ffc9-1        ;36 Chkout
9b8d cb ff                 .wo $ffcc-1        ;38 Clrch
9b8f ce ff                 .wo $ffcf-1        ;40 Basin
9b91 d1 ff                 .wo bsout-1        ;42
```

Listing 6.4.2 (Teil 1)

```
9b93 9a 9b        .wo token-1        ;44
9b95 6a a9        .wo $a96b-1        ;46
9b97 e0 ff        .wo $ffe1-1        ;48 STOP-Taste
9b99 27 ba        .wo $ba28-1        ;50 FAC*(A/Y)
```

Listing 6.4.2 (Teil 2)

Nach den ROM-Befehlen folgte in der Grundversion die Token-Wandelroutine. Auch diese werden wir durch eine neue Routine ersetzen. Dies geschieht, weil sich die Länge unserer Befehlstabelle recht schnell der Grenze von 256 Byte nähert, für welche die Token-Wandelroutine ausgelegt ist. Die neue Version wird diese Beschränkung nicht mehr aufweisen.

Als kleine Besonderheit finden Sie am Anfang der Routine den Befehl 'JSR AUTOEXEC'. Dieser Unterprogrammaufruf wird später für den Befehl 'AUTO' benötigt, und dort auch erklärt.

```
               ;---------------------------------------
               ;---  Eingabe in Token wandeln       ---
               ;---------------------------------------
9b9b c6 01     token:   dec pp             ;Weiter bei der AUTO-Routine
9b9d 20 54 a3           jsr autoexec
9ba0 e6 01              inc pp
9ba2 d0 51              bne nexttk

9ba4 a0 ff     vgl1:    ldy #$ff           ;Y zeigt auf die Befehlsliste
9ba6 a9 2c              lda #<beflist      ;azg & Y auf die Befehlstabelle
9ba8 85 fb              sta azg
9baa a9 9c              lda #>beflist
9bac 85 fc              sta azg+1
9bae a9 01              lda #1             ;Tokennummer ist 1
9bb0 85 fd              sta azg2
9bb2 ca                 dex
9bb3 c8        vgl:     iny                ;naechstes Zeichen vergleichen
9bb4 e8                 inx
9bb5 b1 fb     vgl2:    lda (azg),y        ;Befehlstext mit Zeile vergl.
9bb7 f0 07              beq eob            ;Befehlswort-Ende
```

Listing 6.4.2-1 (Teil 1)

```
9bb9 dd 00 02          cmp zeile,x
9bbc d0 1c             bne ungl         ;nein
9bbe f0 f3             beq vgl
9bc0 a4 02      eob:   ldy xmem         ;Befehl gefunden
9bc2 a9 d2             lda #xtoken      ;XBASIC-Token in Zeile
bringen
9bc4 99 00 02          sta zeile,y
9bc7 c8                iny
9bc8 a5 fd             lda azg2
9bca 99 00 02          sta zeile,y
9bcd c8                iny
9bce bd 00 02   versch: lda zeile,x     ;Rest des Befehlswortes loeschen
9bd1 99 00 02          sta zeile,y
9bd4 f0 4a             beq tonexttk     ;Laenge der Zeile korrigieren
9bd6 e8                inx
9bd7 c8                iny
9bd8 d0 f4             bne versch

9bda a6 02      ungl:  ldx xmem         ;Ende des Befehlswortes in
Tabelle suchen
9bdc 88                dey
9bdd e6 fd             inc azg2         ;Tokennummer +1
9bdf c8         sbe:   iny
9be0 b1 fb             lda (azg),y
9be2 d0 fb             bne sbe
9be4 38                sec
9be5 98                tya
9be6 65 fb             adc azg
9be8 85 fb             sta azg
9bea a9 00             lda #0
9bec a8                tay
9bed 65 fc             adc azg+1
9bef 85 fc             sta azg+1
9bf1 b1 fb             lda (azg),y
9bf3 d0 c0             bne vgl2         ;nein, weitervergleichen

9bf5 e8         nexttk: inx             ;ab naechter Position in Zeile
 neusuchen
9bf6 86 02             stx xmem
9bf8 bd 00 02          lda zeile,x      ;Zeile beendet ?
9bfb f0 1b             beq aborttk
9bfd c9 83             cmp #131         ;DATA-Token ?
9bff f0 17             beq aborttk      ;dann Rest der Zeile nicht
beachten
9c01 c9 8f             cmp #143         ;REM-Token ?
9c03 f0 13             beq aborttk      ;dann Rest nicht beachten
9c05 c9 22             cmp #34          ;String ?
9c07 d0 9b             bne vgl1         ;nein
9c09 e8         hkomma: inx             ;Strings ueberlesen
9c0a bd 00 02          lda zeile,x
9c0d f0 09             beq aborttk      ;Ende der Zeile erreicht
9c0f c9 22             cmp #34
9c11 d0 f6             bne hkomma
9c13 e8                inx
```

Listing 6.4.2-1 (Teil 2)

```
9c14 86 02                   stx xmem
9c16 d0 8c                   bne vgl1          ;Stringende erreicht

9c18 a4 0b      aborttk:     ldy $b            ;Laenge der Zeile laden
9c1a a9 00                   lda #0
9c1c 99 fd 01                sta $200-3,y
9c1f 60                      rts               ;fertig

9c20 98         tonexttk:    tya               ;Laenge der Zeile korrigieren
9c21 a6 02                   ldx xmem          ;Laenge der neuen Zeile
9c23 e8                      inx
9c24 18                      clc               ;+6 fuer Verwaltung
9c25 69 05                   adc #5
9c27 85 0b                   sta $b            ;ergibt die Laenge
9c29 d0 ca                   bne nexttk

```

Listing 6.4.2-1 (Teil 3)

An die Token-Wandeloutine schließt sich die Tabelle der neuen Basic-Befehle, sowie ihrer Adressen an. Diese Tabellen müssen natürlich auch gemäß unserer Befehlstabelle (siehe Kapitel 4/6.4.9) erweitert werden.

Beachten Sie, daß für den Befehl mit der Nummer 32 ein Dummy-Eintrag vorgenommen wurde. Dieses Vorgehen ist nötig, da die CHRGET-Routine, welche das nächste Zeichen aus dem Programm holt, Leerzeichen überliest. Da Leerzeichen den Code 32 haben, wird der Befehl mit der Nummer 32 durch einen doppelten Eintrag (hier: 'dir') gesperrt.

```
                ;--------------------------------------
                ;---  Liste der XBASIC Befehle       ---
                ;--------------------------------------
                ;  1. Liste der Funktionen
9c2b 0e         funktionen:.by 13+1            ;Anzahl der Funktionen +1

9c2c b8 28 00   beflist:     .by 184,'(,0      ;FRE(  Liste der Funktionsworte
9c2f 41 54 28                .by "at(",0       ;AT(
00
9c33 48 45 58                .by "hex$",0      ;HEX$
```

Listing 6.4.2-2 (Teil 1)

```
24 00
9c38 44 45 43        .by "dec",0          ;DEC
00
9c3c 49 4e 4b        .by "inkey$",0       ;INKEY$
45 59 24 00
9c43 49 4e 53        .by "instr(",0       ;INSTR(
54 52 28 00
9c4a 53 54 52        .by "string$(",0     ;STRING$
49 4e 47 24 28 00
9c53 53 50 41        .by "space$(",0      ;SPACE$
43 45 24 28 00
9c5b 44 52 45        .by "dreek",0        ;DREEK
45 4b 00
9c61 52 45 45        .by "reek",0         ;REEK
4b 00
9c66 44 45 45        .by "deek",0         ;DEEK
4b 00
9c6b 53 54 24        .by "st$",0          ;ST$
00
9c6f 54 49 4d        .by "time",0         ;TIME
45 00

              ;  2. Liste der Befehle
9c74 43 4c 53        .by "cls",0          ;CLS  Liste der Befehlsworte
00
9c78 4f 4c 44        .by "oldchar",0      ;OLDCHAR
43 48 41 52 00
9c80 43 36 34        .by "c64",0          ;C64
00
9c84 58 42 41        .by "xbasic",0       ;XBASIC
53 49 43 00
9c8b 49 4e 53        .by "inst",0         ;INST
54 00
9c90 96 43 48        .by 150,"char",0     ;DEFCHAR
41 52 00
9c96 4f 4c 44        .by "old",0          ;OLD
00
9c9a 44 52 4f        .by "droke",0        ;DROKE
4b 45 00
9ca0 52 4f 4b        .by "roke",0         ;ROKE
45 00
9ca5 44 4f 4b        .by "doke",0         ;DOKE
45 00
9caa 41 55 a4        .by "au",164,0       ;AUTO
00
9cae 53 57 41        .by "swap",0         ;SWAP
50 00
9cb3 44 49 52        .by "dir",0          ;DIR
00
9cb7 51 93 00        .by 'q,147,0         ;QLOAD
9cba 43 4f 4c        .by "collect",0      ;COLLECT
4c 45 43 54 00
9cc2 48 45 41        .by "header",0       ;HEADER
```

Listing 6.4.2-2 (Teil 2)

```
44 45 52 00
9cc9 52 45 4e           .by "rename",0       ;RENAME
41 4d 45 00
9cd0 53 43 52           .by "scratch",0      ;SCRATCH
41 54 43 48 00
9cd8 44 49 52           .by "dir",0          ;Dummy fuer Befehl 32
00
9cdc 49 4e 49           .by "initialise",0 ;INITIALISE
54 49 41 4c 49 53 45 00
9ce7 53 45 54           .by "setd",193,'r,0;SETDATNR
44 c1 52 00
9cee 50 41 55           .by "pause",0        ;PAUSE
53 45 00
9cf4 53 45 54           .by "settime",0      ;SETTIME
54 49 4d 45 00
9cfc 53 45 54           .by "seteol",0       ;SETEOL
45 4f 4c 00
9d03 4c 49 4e           .by "line",133,0     ;LINEINPUT
45 85 00
9d09 4c 49 4e           .by "line",132,0     ;LINEINPUT#
45 84 00

9d0f 00                 .by 0                ;Tabellenabschluss

9d10 87 48 00   fk100:  .by $87,$48,0,0,0  ;Fliesskommakonstante 100
00 00

        RESERVE2:  .db 244+beflist-RESERVE2;Reserve fuer Erweiterungen


        ;Tabelle der Adressen der Befehle & Funktionen
9d20 f6 9f  befehle:  .wo BBfre-1      ;FRE(  * Funktionen
9d22 e4 a0            .wo BBat-1       ;AT(
9d24 81 a0            .wo BBhex-1      ;HEX$
9d26 29 a0            .wo BBdez-1      ;DEC
9d28 ba a0            .wo BBinkey-1    ;INKEY$
9d2a 98 a1            .wo BBinstr-1    ;INSTR
9d2c 31 a1            .wo BBstring-1   ;STRING$
9d2e 15 a1            .wo BBspaces-1   ;SPACES
9d30 fc a3            .wo BBdreek-1    ;DREEK
9d32 1c a4            .wo BBreek-1     ;REEK
9d34 39 a4            .wo BBdeek-1     ;DEEK
9d36 13 a5            .wo BBst-1       ;ST$
9d38 46 a6            .wo BBtime-1     ;TIME

9d3a 43 e5            .wo clrscr-1     ;CLS   * Befehle
9d3c 63 9a            .wo setzeichen-1 ;OLDCHAR
9d3e e1 fc            .wo reset-1      ;C64
9d40 ff 99            .wo BBxbasic-1   ;XBASIC
9d42 aa 9e            .wo BBinst-1     ;INST
9d44 18 9f            .wo BBdefchar-1  ;DEFCHAR
9d46 91 9e            .wo BBold-1      ;OLDCHAR
9d48 d5 a3            .wo BBdroke-1    ;DROKE
```

Listing 6.4.2-2 (Teil 3)

```
9d4a 98 a3                  .wo BBroke-1        ;ROKE
9d4c ac a3                  .wo BBdoke-1        ;DOKE
9d4e 3d a3                  .wo BBauto-1        ;AUTO
9d50 59 a4                  .wo BBswap-1        ;SWAP
9d52 ab a4                  .wo BBdir-1         ;DIR
9d54 23 a2                  .wo BBqload-1       ;QLOAD
9d56 48 a5                  .wo BBcollect-1     ;COLLECT
9d58 7a a5                  .wo BBheader-1      ;HEADER
9d5a a0 a5                  .wo BBrename-1      ;RENAME
9d5c 7d a5                  .wo BBscratch-1     ;SCRATCH
9d5e 00 00                  .wo 0               ;Dummy fuer Befehl 32
9d60 71 a5                  .wo BBinitial-1     ;INITIALISE
9d62 a3 a4                  .wo BBsetdatnr-1    ;SETDATNR
9d64 f9 a5                  .wo BBpause-1       ;PAUSE
9d66 af a6                  .wo BBsettime-1     ;SETCLOCK
9d68 3f a7                  .wo BBseteol-1      ;SETEOL
9d6a 47 a7                  .wo BBlineinput-1   ;LINEINPUT
9d6c 96 a7                  .wo BBlineindev-1   ;LINEINPUT#

            RESERVE3:       .db 100+befehle-RESERVE3;Reserve fuer Erweiterungen
```

Listing 6.4.2-2 (Teil 4)

Nun müssen wir nur noch in der Einschaltmeldung die Versionsnummer von '1.0' auf '1.1' ändern. Dann können wir uns den Befehlsroutinen zuwenden.

```
                 ;---------------------------------------
                 ;---  XBASIC   Einschaltmeldung      ---
                 ;---------------------------------------
9de2 93 d8 c2    meldung:   .by 147,"XBASIC Version 1.1",13
c1 d3 c9 c3 20 d6 45 52 53 49 4f 4e 20 31 2e 31 0d
9df6 28 43 29               .by "(c) M.Friese 1987",13,13,0
20 cd 2e c6 52 49 45 53 45 20 31 39 38 37 0d 0d 00
```

Listing 6.4.2-3

4/6.4.2.11

QLOAD FILE$,DEV

Als ersten Befehl der Version 1.1 werden wir den Befehl QLOAD implementieren. Dieser Befehl wird uns dabei helfen, ein kleines Problem zu bewältigen.

In der Version 1.1 ist die Tabelle der Befehle notwendigerweise geändert worden. Da auch neue Funktionen hinzugekommen sind, und alle Funktionen vor den Befehlen stehen müssen, haben einige der Befehle aus der Grundversion neue Nummern erhalten. Laden Sie nun ein Programm, welches mit der Grundversion erstellt wurde, mit 'LOAD' in den Speicher, so ist es unter Umständen nicht ohne Änderungen lauffähig.

QLOAD soll dieses Problem beseitigen. Im Gegensatz zum LOAD-Befehl lädt QLOAD eine ASCII-Datei zeilenweise ein. Bei jeder Zeile wird getestet, ob sie mit einer Zahl beginnt. Ist dies der Fall, wird sie als Zeilennummer angesehen, der Text mit Hilfe der Token-Wandelroutine in die interne Darstellung gewandelt, und die Zeile im Speicher abgelegt. Alle anderen Zeilen werden überlesen.

Um nun ein altes Programm zu übertragen, muß es zunächst als ASCII-Datei auf Diskette geschrieben werden. Dazu wird die Grundversion von XBASIC geladen und gestartet. Dann wird das Programm mit 'LOAD' geladen. Mit Hilfe der folgenden Zeilen wird es als ASCII-Datei auf Diskette geschrieben:

```
OPEN 2,8,2,"dateiname,p,w":CMD2:LIST
CLOSE2
```

Mit Hilfe des Befehls QLOAD"dateiname",8 oder QLOAD"dateiname" kann diese ASCII-Datei von der Diskette als Programmdatei eingelesen werden. Wird ein anderes Gerät als die Diskettenstation (Gerätenummer 8) verwendet, muß sie wie bei dem LOAD-Befehl angegeben werden. Fehlt die Nummer, wird 8 angenommen.

Zu Beachten ist aber, daß der Puffer zur Wandlung in die interne Darstellung maximal 88 Zeichen faßt. Längere Zeilen, welche mit Hilfe von Abkürzungen erzeugt wurden, werden überlesen! Die Zeilennummern dieser Zeilen werden aber auf dem Bildschirm angezeigt.

Außerdem ist zu beachten, daß QLOAD nur im Direktmodus arbeitet. Wird der Befehl von einem Programm aus aufgerufen, so erfolgt die Fehlermeldung "UNDEF'D FUNCTION".

```

                ;---------------------------------------
                ;--- XBASIC-Befehl  QLOAD DATEI$,DEV ---
                ;---------------------------------------
                status:    .eq $90          ;Diskettenstatus

a224 a6 3a      BBqload:   ldx $3a          ;Test auf Direkt-Modus
a226 e8                    inx
a227 f0 05                 beq qldirekt
a229 a2 1b                 ldx #27          ;"undef'd function"
a22b 4c 06 9b              jmp fehler
a22e a5 2b      qldirekt:  lda 43           ;$ae auf Programmstart
a230 85 ae                 sta $ae
a232 a5 2c                 lda 44
a234 85 af                 sta $af
a236 20 0b 9b              jsr frmevl       ;String DATEI$ holen
a239 a2 04                 ldx #4
a23b 20 b6 9a              jsr basicrom
a23e 98                    tya              ;Laenge in Accu
a23f a6 22                 ldx $22          ;Als Filenamen setzen
a241 a4 23                 ldy $23
a243 20 06 a3              jsr filename
a246 20 79 00              jsr chrg
op     ;Datei lesen
a25d a0 ff      qloadline: ldy #$ff         ;eine Zeile lesen
a25f c8         qloadl:    iny
a260 c0 58                 cpy #88          ;Zeile zu lang?
a262 f0 76                 beq zulang
a264 20 fc a2              jsr basin        ;Zeichen lesen
a267 99 00 02              sta $200,y       ;..speichern
a26a c9 0d                 cmp #13          ;bis <RETURN>
a26c d0 f1                 bne qloadl
a26e ad 00 02              lda $200         ;Zeile beginnt mit Ziffer ?
a271 c9 30                 cmp #'0
a273 90 49                 bcc unload       ;nein
a275 c9 3a                 cmp #'9+1
a277 b0 45                 bcs unload       ;nein
a279 a9 00                 lda #0
a27b 99 00 02              sta $200,y       ;Puffer mit 0 abschliessen
a27e a9 ff                 lda #<$200-1
a280 85 7a                 sta $7a
```

Listing 6.4.2.11 (Teil 1)

```
a282 a9 01                  lda #>$200-1
a284 85 7b                  sta $7b
a286 20 73 00               jsr chrget
a289 a2 2e                  ldx #46             ;Zeilennummer wandeln
a28b 20 b6 9a               jsr basicrom
a28e a2 2c                  ldx #44             ;Eingabe in Token wandeln
a290 20 b6 9a               jsr basicrom
a293 a0 00                  ldy #0
a295 a9 08                  lda #8              ;Link-Pointer setzen
a297 91 ae                  sta ($ae),y
a299 c8                     iny
a29a 91 ae                  sta ($ae),y
a29c c8                     iny
a29d a5 14                  lda $14             ;Zeilennummer setzen
a29f 91 ae                  sta ($ae),y
a2a1 c8                     iny
a2a2 a5 15                  lda $15
a2a4 91 ae                  sta ($ae),y
a2a6 c8                     iny
a2a7 a2 00                  ldx #0
a2a9 bd 00 02     puttheline:lda $200,x
a2ac 91 ae                  sta ($ae),y
a2ae e8                     inx
a2af c8                     iny
a2b0 c9 00                  cmp #0
a2b2 d0 f5                  bne puttheline
a2b4 98                     tya
a2b5 18                     clc
a2b6 65 ae                  adc $ae
a2b8 85 ae                  sta $ae
a2ba 90 02                  bcc unload
a2bc e6 af                  inc $af
a2be a5 90        unload:   lda status          ;Datei beendet ?
a2c0 f0 9b                  beq qloadline       ;nein, naechste Zeile
a2c2 a9 00                  lda #0
a2c4 91 ae                  sta ($ae),y
a2c6 c8                     iny
a2c7 91 ae                  sta ($ae),y
a2c9 a8                     tay
a2ca 91 7a                  sta ($7a),y
a2cc c8                     iny
a2cd c8                     iny
a2ce 91 7a                  sta ($7a),y
a2d0 20 31 a3               jsr readfileclo     ;Datei schliessen
a2d3 a9 00                  lda #0              ;ggf. AUTO Zeilennummer loeschen
a2d5 85 c6                  sta 198
a2d7 4c 92 9e               jmp BBold
a2da a0 00        zulang:   ldy #0              ;Zeile zu lang-> Zeilennummer
ausgeben
a2dc b9 00 02     zula:     lda $200,y
a2df c9 30                  cmp #'0
```

Listing 6.4.2.11 (Teil 2)

```
a2e1 90 0a               bcc zlende
a2e3 c9 39               cmp #'9
a2e5 b0 06               bcs zlende
a2e7 20 01 a3            jsr basout      ;Zeilennummer ausgeben
a2ea c8                  iny
a2eb d0 ef               bne zula
a2ed 20 fc a2  zlende:   jsr basin       ;bis Zeilenende lesen
a2f0 c9 0d               cmp #13
a2f2 d0 f9               bne zlende
a2f4 a9 2c               lda #',         ;Komma ausgeben
a2f6 20 01 a3            jsr basout
a2f9 4c be a2            jmp unload
```

Listing 6.4.2.11 (Teil 3)

QLOAD verwendet einige Unterprogramme, welche in den folgenden Zeilen definiert werden, und später auch von anderen Befehle genutzt werden.

Das Unterprogramm <BASIN> liest ein Zeichen in den Akku ein, und <BASOUT> gibt das Zeichen im Akku aus. <FILENAME> setzt den Dateinamen. Dabei muß im Akku die Länge des Namens, und in X/Y die Adresse stehen.

Das Unterprogramm <READFILEOP> eröffnet eine Datei mit der logischen Dateinummer 80 (kann geändert werden: siehe SETDATNR) zum Lesen auf der Diskette. Außerdem lenkt es die Eingaberoutine <BASIN> auf diese Datei um. Die Datei kann durch <READFILECLO> wieder geschlossen werden. Die Eingabe erfolgt dann wieder von der Tastatur.

```
a2fc a2 28     basin:     ldx #40        ;ein Zeichen einlesen
a2fe 4c b6 9a             jmp basicrom

a301 a2 2a     basout:    ldx #42        ;eiu Zeichen ausgeben
a303 4c b6 9a             jmp basicrom

a306 86 02     filename:  stx xmem       ;Filenamen setzen
a308 a2 1c                ldx #28
a30a 4c b6 9a             jmp basicrom
```

Listing 6.4.2.11-1 (Teil 1)

```
a30d 86 02     readfileop:stx xmem          ;Fileparamuter setzen
a30f a2 1a                ldx #26
a311 20 b6 9a             jsr basicrom
a314 a2 1e                ldx #30           ;OPEN
a316 20 b6 9a             jsr basicrom
a319 b0 0d                bcs openerr       ;Fehler
a31b ad ab a4             lda dateinr
a31e 85 02                sta xmem
a320 a2 22                ldx #34           ;& CHKIN
a322 20 b6 9a             jsr basicrom
a325 b0 01                bcs openerr
a327 60                   rts
a328 48        openerr:   pha               ;Fehler melden
a329 20 31 a3             jsr readfileclo
a32c 68                   pla
a32d aa                   tax
a32e 4c 06 9b             jmp fehler

a331 ad ab a4  readfileclo:lda dateinr      ;CLOSE dateinnr
a334 a2 20                ldx #32
a336 20 b6 9a             jsr basicrom
a339 a2 26                ldx #38           ;Eingabe wieder normal
a33b 4c b6 9a             jmp basicrom
```

Listing 6.4.2.11-1 (Teil 1)

4/6.4.2.12

AUTO STEP

Der AUTO-Befehl gibt nach jeder Zeileneingabe die nächste Zeilennummer vor. Dies ist vor allem zum Eintippen längerer Programme recht praktisch.

Nach dem Starten der Basic-Erweiterung ist der AUTO-Befehl aktiv. Er testet bei jeder Eingabe, ob sie mit einer Zeilennummer beginnt. Ist dies der Fall, wird die Zeilennummer um 10 erhöht, und in der nächsten Zeile als Zeilennummer vorgegeben werden. Bei der Direkteingabe von Befehlen, wird so die Ausgabe von Zeilennummern durch den AUTO-Befehl vermieden.

Ist eine andere Schrittweite gewünscht, so kann die Schrittweite durch Eingabe von 'AUTO STEP' geändert werden. Für die Schrittweite 100 gibt man also 'AUTO 100' ein.

Der AUTO-Befehl wird durch die Eingabe 'AUTO 0' abgeschaltet. Jede andere Schrittweite schaltet ihn wieder ein.

```
                 ;---------------------------------------
                 ;--- XBASIC-Befehl  AUTO STEP        ---
                 ;---------------------------------------
a33e 20 3b 9b    BBauto:    jsr getadr       ;Schrittweite holen
a341 a5 14                  lda $14          ;Schrittweite speichern
a343 8d 52 a3               sta autostep
a346 a5 15                  lda $15
a348 8d 53 a3               sta autostep+1   ;Flag fuer 'AUTO an'
a34b 05 14                  ora $14          ;setzen bzw. loeschen
a34d 8d 51 a3               sta autoon
a350 60                     rts

a351 01          autoon:    .by 1            ;Flag fuer AUTO an
a352 0a 00       autostep:  .wo 10           ;Schrittweite fuer AUTO
```

Listing 6.4.2.12

Erreicht wird die Ausgabe der nächsten Zeilennummer durch die Routine <AUTOEXEC>, welche von der Token-Wandelroutine aufgerufen wird, und so bei jeder Eingabe durchlaufen wird. Sie erledigt die gesamte Arbeit. Der AUTO-Befehl selbst legt nur die Schrittweite im Speicher ab, und setzt eine Flagge für AUTO ein- bzw. ausgeschaltet.

```
a354 ad 00 02    autoexec:  lda $200         ;erstes Zeichen merken
a357 48                     pha
a358 a2 18                  ldx #24
a35a 20 b6 9a               jsr basicrom     ;Wandeln der normalen C64 Befehle
a35d 84 0b                  sty $b           ;Zeilenlaenge merken
a35f 68                     pla
a360 ae 51 a3               ldx autoon       ;AUTO an ?
```

Listing 6.4.2.12-1 (Teil 1)

```
a363 f0 31                   beq nonauto         ;nein
a365 c9 30                   cmp #'0
a367 90 2d                   bcc nonauto         ;ja
a369 c9 3a                   cmp #'9+1
a36b b0 29                   bcs nonauto
a36d 18                      clc                 ;Zeilennummer merken & neue
berechnen
a36e a5 14                   lda $14
a370 48                      pha
a371 6d 52 a3                adc autostep
a374 a8                      tay
a375 a5 15                   lda $15
a377 48                      pha
a378 6d 53 a3                adc autostep+1
a37b 20 18 a0                jsr ergebnis        ;in Fliesskomma wandeln
a37e a2 16                   ldx #22             ;in ASCII wandeln
a380 20 b6 9a                jsr basicrom
a383 a2 ff                   ldx ##ff            ;String in Tastaturpuffer
a385 e8           putintast: inx
a386 bd 01 01                lda $101,x
a389 9d 77 02                sta $277,x
a38c d0 f7                   bne putintast
a38e 86 c6                   stx $c6             ;Stringlaenge als Zeichen im
Puffer speichern
a390 68                      pla                 ;Zeilennummer zurueckholen
a391 85 15                   sta $15
a393 68                      pla
a394 85 14                   sta $14
a396 a2 ff        nonauto:   ldx ##ff            ;Weiter in der Tokenroutine
a398 60                      rts
```

Listing 6.4.2.12-1 (Teil 2)

4/6.4.2.13

ROKE AD,W

Der Befehl ROKE arbeitet genau wie der POKE-Befehl. Im Unterschied zum POKE-Befehl wird jedoch vor der Ausführung der gesamte Speicher auf RAM geschaltet. So können Werte auch in den RAM-Bereich von $D000 bis $DFFF abgelegt werden.

```
                ;-----------------------------------------
                ;--- XBASIC-Befehl  ROKE AD,W          ---
                ;-----------------------------------------
a399 20 46 9b   BBroke:     jsr getadby        ;Adresse und Wert holen
a39c a0 00                  ldy #0
a39e a5 01                  lda pp             ;Speicheraufteilung retten
a3a0 48                     pha
a3a1 78                     sei                ;Alles auf RAM
a3a2 a9 22                  lda #34
a3a4 85 01                  sta pp
a3a6 8a                     txa                ;Wert speichern
a3a7 91 14                  sta ($14),y
a3a9 68                     pla                ;Speicher wieder normal
a3aa 85 01                  sta pp
a3ac 60                     rts
```

Listing 6.4.2.13

4/6.4.2.14

DOKE AD,W

Der Befehl DOKE arbeitet wie der POKE-Befehl. Er ermöglicht jedoch 16-Bit Werte gemäß der 6502-Reihenfolge in zwei aufeinanderfolgende Speicherstellen abzulegen. Eine Befehlsfolge der Form:

```
POKE 251,100:POKE252,123
```

kann also durch DOKE 251,31588 ersetzt werden (wegen 100+256*123=31588).

```
                 ;--------------------------------------
                 ;--- XBASIC-Befehl  DOKE AD,W       ---
                 ;--------------------------------------
a3ad 20 3b 9b    BBdoke:    jsr getadr        ;Adresse holen
a3b0 a5 14                  lda $14           ;und in azg bringen
a3b2 a6 15                  ldx $15
a3b4 85 fb                  sta azg
a3b6 86 fc                  stx azg+1
a3b8 20 23 9b               jsr chkkom        ;16Bit Wert holen
a3bb 20 3b 9b               jsr getadr
a3be a9 91                  lda #$91          ;LDA-Befehl in RAM-Routine in
a3c0 8d d9 9a               sta getazg+2      ;STA aendern
a3c3 a0 00                  ldy #0            ;erstes Byte setzen
a3c5 a5 14                  lda $14
a3c7 20 d7 9a               jsr getazg
a3ca c8                     iny               ;zweites Byte setzen
a3cb a5 15                  lda $15
a3cd 20 d7 9a               jsr getazg
a3d0 a9 b1                  lda #$b1          ;Routine wieder normal
a3d2 8d d9 9a               sta getazg+2
a3d5 60                     rts
```

Listing 6.4.2.14

4/6.4.2.15

DROKE AD,W

Der Befehl DROKE arbeitet wie der DOKE-Befehl. Im Unterschied zum DOKE-Befehl wird jedoch vor der Ausführung der gesamte Speicher auf RAM geschaltet. So können 16-Bit Werte auch in den RAM-Bereich von $D000bis $DFFF abgelegt werden.

```
               ;---------------------------------------
               ;--- XBASIC-Befehl  DROKE AD,W       ---
               ;---------------------------------------
a3d6 20 3b 9b  BBdroke:   jsr getadr         ;Adresse holen
a3d9 a5 14                lda $14            ;in azg bringen
a3db a6 15                ldx $15
a3dd 85 fb                sta azg
a3df 86 fc                stx azg+1
a3e1 20 23 9b             jsr chkkom         ;Wert lesen
a3e4 20 3b 9b             jsr getadr
a3e7 78                   sei                ;alles auf RAM
a3e8 a9 22                lda #34
a3ea 85 01                sta pp
a3ec a5 14                lda $14            ;erstes Byte speichern
a3ee a0 00                ldy #0
a3f0 91 fb                sta (azg),y
a3f2 c8                   iny
a3f3 a5 15                lda $15            ;zweites Byte speichern
a3f5 91 fb                sta (azg),y
a3f7 a9 36                lda ##$36          ;Speicher normal
a3f9 85 01                sta pp
a3fb 58                   cli
a3fc 60                   rts
```

Listing 6.4.2.15

4/6.4.2.16

A=DREEK(AD)

So wie PEEK die Umkehrung von POKE ist, so ist DREEK die Umkehrung von DROKE. DREEK schaltet den gesamten Speicher auf RAM, und liest dann einen 16-Bit Wert aus dem Speicher.

```
                ;--------------------------------------
                ;--- XBASIC-Befehl  A=DREEK(ADR)     ---
                ;--------------------------------------
a3fd 20 73 00   BBdreek:  jsr chrget
a400 20 5e 9b             jsr klammer        ;Klammerausdruck holen
a403 20 2b 9b             jsr adresse        ;in pos. Integer wandeln
a406 78                   sei                ;alles auf RAM schalten
a407 a0 01                ldy #1
a409 a9 34                lda #$34
a40b 85 01                sta pp
a40d b1 14                lda ($14),y        ;zweites Byte holen
a40f aa                   tax                ;und merken
a410 88                   dey                ;erstes Byte holen
a411 b1 14                lda ($14),y
a413 a8                   tay                ;in Y bringen
a414 a9 36                lda #$36           ;Speicher normal
a416 85 01                sta pp
a418 58                   cli
a419 8a                   txa                ;Wert in Accu/Y liefern
a41a 4c 18 a0             jmp ergebnis
```

Listing 6.4.2.16

4/6.4.2.17

A=REEK(AD)

REEK arbeitet wie Peek. Vor dem Lesen des Wertes wird jedoch der gesamten Speicher auf RAM geschaltet.

```
                ;---------------------------------------
                ;--- XBASIC-Befehl  A=REEK(ADR)      ---
                ;---------------------------------------
a41d 20 73 00   BBreek:   jsr chrget
a420 20 5e 9b             jsr klammer      ;Klammerausdruck holen
a423 20 2b 9b             jsr adresse      ;in pos. Integer wandeln
a426 78                   sei              ;alles auf RAM schalten
a427 a0 00                ldy #0
a429 a9 34                lda #$34
a42b 85 01                sta pp
a42d b1 14                lda ($14),y      ;Byte holen
a42f a8                   tay              ;und merken
a430 a9 36                lda #$36         ;Speicher normal
a432 85 01                sta pp
a434 58                   cli
a435 a9 00                lda #0           ;Wert in Accu/Y liefern
a437 4c 18 a0             jmp ergebnis
```

Listing 4/6.4.2.17

4/6.4.2.18

A=DEEK(AD)

DEEK arbeitet wie PEEK, jedoch mit 16-Bit-Werten. Mit Hilfe der Befehle PEEK, POKE, DEEK, DOKE, REEK, ROKE, DREEK und DROKE kann man im gesamten Speicher die Speicherzellen lesen oder beschreiben. Die folgende Tabelle soll über diese Befehle noch einmal eine Übersicht geben:

	Speicher normal		Speicher nur RAM	
	Lesen	Speichern	Lesen	Speichern
8-Bit	PEEK	POKE	REEK	ROKE
16-Bit	DEEK	DOKE	DREEK	DROKE

```
                 ;--------------------------------------
                 ;--- XBASIC-Befehl  A=DEEK(ADR)     ---
                 ;--------------------------------------
a43a 20 73 00    BBdeek:    jsr chrget
a43d 20 5e 9b               jsr klammer       ;Klammerausdruck holen
a440 20 2b 9b               jsr adresse       ;in pos. Integer wandeln
a443 a5 14                  lda $14           ;Adressein azg bringen
a445 85 fb                  sta azg
a447 a5 15                  lda $15
a449 85 fc                  sta azg+1
a44b a0 01                  ldy #1
a44d 20 d7 9a               jsr getazg        ;zweites Byte holen
a450 aa                     tax               ;und merken
a451 88                     dey               ;erstes Byte holen
a452 20 d7 9a               jsr getazg
a455 a8                     tay               ;in Y bringen
a456 8a                     txa               ;Wert in Accu/Y liefern
a457 4c 18 a0               jmp ergebnis
```

Listing 4/6.4.2.18

4/6.4.2.19

SWAP A, B

Der Befehl SWAP vertauscht die Variableninhalte der beiden nachfolgenden Variablen. Dabei arbeitet er mit jedem Variablentyp zusammen, sowohl als einfache Variable, als auch mit Variablen aus ARRAY's. Die einzige Bedingung ist, daß die Variablen vom gleichen Typ sind. Zulässige Eingaben sind also:

SWAP A, B : SWAP A%,B% : SWAP A$, B$
SWAP A,A(8) : SWAP A$ (12),B$

nicht aber:

SWAP A%,B : SWAP A$,B% oder ähnliche.

Einfache Variablen werden grundsätzlich in sieben Byte gespeichert, wobei die ersten beiden Byte den Variablennamen und den Typ enthalten. Die folgenden Byte enthalten den Variablenwert. Überflüssige Byte werden mit Null aufgefüllt. In Array's werden jedoch nur die benötigten Bytes angelegt. Die sind bei REAL-Typen 5 Byte, bei Integer zwei Byte, und bei Strings drei Byte.

Um nicht zwischen Array's und normalen Variablen unterscheiden zu müssen, vertauscht SWAP grundsätzlich nur die tatsächlich genutzten Bytes. So arbeitet er problemlos auch mit gemischten Angaben.

```
                  ;-------------------------------------------
                  ;--- XBASIC-Befehl  SWAP A,B            ---
                  ;-------------------------------------------
a45a a2 14        BBswap:   ldx #20            ;Variable suchen
a45c 20 b6 9a               jsr basicrom
a45f a5 0d                  lda $d             ;String-Flag
a461 85 fd                  sta azg2
a463 a5 0e                  lda $e             ;Integer-Flag
a465 85 fe                  sta azg2+1
a467 a5 47                  lda $47            ;Variablenadrese in azg merken
a469 85 fb                  sta azg
a46b a5 48                  lda $48
```

Listing 4/6.4.2.19 (Teil1)

```
a46d 85 fc                        sta azg+1
a46f 20 23 9b                     jsr chkkom        ;zweite Variable suchen
a472 a2 14                        ldx #20
a474 20 b6 9a                     jsr basicrom      ;varsuch
a477 a5 0d                        lda $d            ;String-Flag vergleichen
a479 c5 fd                        cmp azg2
a47b d0 06                        bne typerr2
a47d a5 0e                        lda $e            ;Integer-Flag vergleichen
a47f c5 fe                        cmp azg2+1
a481 f0 03                        beq typok
a483 4c 11 9f     typerr2:        jmp typerr        ;Typen verschieden
a486 a5 0d        typok:          lda $d            ;String ?
a488 d0 07                        bne swapstring
a48a a5 0e                        lda $e            ;Integer ?
a48c d0 06                        bne swapint
a48e a0 04                        ldy #4            ;5 Byte fuer REAL
a490 2c                           bit
a491 a0 02        swapstring:ldy #2                 ;3 Byte fuer String
a493 2c                           bit
a494 a0 01        swapint:        ldy #1            ;2 Byte fuer Integer
a496 b1 fb        swapit:         lda (azg),y       ;Bytes vertauschen
a498 aa                           tax
a499 b1 47                        lda ($47),y
a49b 91 fb                        sta (azg),y
a49d 8a                           txa
a49e 91 47                        sta ($47),y
a4a0 88                           dey               ;bis alle Bytes getauscht
a4a1 10 f3                        bpl swapit
a4a3 60                           rts
```

Listing 4/6.4.2.19 (Teil 2)

4/6.4.2.20

SETDATNR X

Wie bei dem Befehl QLOAD bereits erwähnt wurde, arbeiten verschiedene Befehle mit der OPEN-Routine des Betriebssystems. Diese benötigt eine logische Dateinummer, welche festgelegt werden muß. Für die XBASIC-Version 1.1 wurde die Nummer 80 gewählt, da diese praktisch nie benutzt wird. Sollte dennoch einmal ein Konflikt mit einem Programm auftreten, so kann man die interne Dateinummer auch ändern, indem man den Befehl 'SETDATNR Dateinummer' eingibt.

```
                ;---------------------------------------
                ;--- XBASIC-Befehl  SETDATNR X       ---
                ;---------------------------------------
                dateinrst: .eq 80           ;Standard Dateinummer fuer
interne Funktionen

a4a4 20 33 9b   BBsetdatnr:jsr getbyte      ;Dateinummer holen
a4a7 8e ab a4              stx dateinr
a4aa 60                    rts

a4ab 50         dateinr:   .by dateinrst    ;log. Dateinummer

```

Listing 4/6.4.2.20

4/6.4.2.21

DIR MASKE$

Der DIR-Befehl zeigt das Inhaltsverzeichnis der Diskette an. Erreicht wird dies, indem der Diskettenstation signalisiert wird, daß man das Inhaltsverzeichnis in den Speicher laden möchte. Statt den Inhalt zu laden, wird er jedoch nur eingelesen und angezeigt.

Als Dateiname wird der String MASKE$ angegeben, so daß man sich gezielte Informationen holen kann:

Beispiel:

DIR"$"	zeigt das gesamte Inhaltsverzeichnis
DIR"$*=P"	zeigt alle Programmdateien
DIR"$*=S"	zeigt sequentiellen Dateien
DIR"$T*"	zeigt alle Dateien, die mit 'T' beginnen.
DIR"$ES*=U"	zeigt alle Dateien, die mit 'ES' beginnen und vom Typ 'USR' sind.

```
                 ;---------------------------------------
                 ;--- XBASIC-Befehl  DIR MASKE$       ---
                 ;---------------------------------------
a4ac 20 0b 9b    BBdir:      jsr frmevl        ;String MASKE$ holen
a4af a2 04                   ldx #4
a4b1 20 b6 9a                jsr basicrom
a4b4 98                      tya               ;Laenge in Acuu
a4b5 a6 22                   ldx $22           ;Als Filenamen setzen
a4b7 a4 23                   ldy $23
a4b9 20 06 a3                jsr filename
a4bc ad ab a4                lda dateinr       ;OPEN dateinr,8,0,MASKE$
a4bf a2 08                   ldx #8
a4c1 a0 00                   ldy #0
a4c3 20 0d a3                jsr readfileop
a4c6 20 fc a2                jsr basin         ;Adresse ueberlesen
a4c9 20 fc a2                jsr basin
a4cc 20 fc a2    dirloop:    jsr basin         ;Link-Pointer ueberlesen
a4cf 20 fc a2                jsr basin
a4d2 20 fc a2                jsr basin         ;Zeilennummer lesen
a4d5 a8                      tay
a4d6 20 fc a2                jsr basin
a4d9 a6 90                   ldx status        ;Diskstatus
a4db d0 1e                   bne dirende
a4dd 20 fe a4                jsr asciiout      ;Zeilennummer ausgeben
a4e0 a9 20                   lda #32           ;Leerzeichen ausgeben
a4e2 20 01 a3                jsr basout
a4e5 20 fc a2    dirloop2:   jsr basin
a4e8 c9 00                   cmp #0
a4ea f0 06                   beq direol        ;Zeilenende
a4ec 20 01 a3                jsr basout
a4ef 4c e5 a4                jmp dirloop2
a4f2 a9 0d       direol:     lda #13           ;neue Zeile
a4f4 20 01 a3                jsr basout
a4f7 a5 90                   lda status        ;Diskettenstatus
a4f9 f0 d1                   beq dirloop       ;weiter
a4fb 4c 31 a3    dirende:    jmp readfileclo   ;CLOSE dateinr

a4fe 20 18 a0    asciiout:   jsr ergebnis      ;in Fliesskomma wandeln
a501 a2 16                   ldx #22           ;in ASCII wandeln
a503 20 b6 9a                jsr basicrom
a506 a0 00                   ldy #0            ;String ausgeben
a508 b9 01 01    asciioutput:lda $101,y
a50b f0 06                   beq asciiend
a50d 20 01 a3                jsr basout
a510 c8                      iny
a511 d0 f5                   bne asciioutput
a513 60          asciiend:   rts
```

Listing 4/6.4.2.21

4/6.4.2.22

A$=ST$

Mit Hilfe des Befehls ST$ kann man den Fehlerkanal der Floppy auslesen. Der Kanal 15 der Diskettenstation wird geöffnet, und die Zeichenkette ausgelesen. Die Zeichen werden zunächst in unseren Pufferbereich <mem> zwischengespeichert, damit die Länge der Zeichenkette festgestellt werden kann. Dann wird ein String der nun bekannten Länge angelegt, und die Zeichen in den String geschrieben.

```
               ;----------------------------------------
               ;--- XBASIC-Befehl  A$=ST$           ---
               ;----------------------------------------
a514 a9 00     BBst:     lda #0
a516 20 06 a3            jsr filename        ;Dateinamen setzen
a519 ad ab a4            lda dateinr         ;OPEN dateinr,8,15
a51c a2 08               ldx #8
a51e a0 0f               ldy #15
a520 20 0d a3            jsr readfileop
a523 a0 ff               ldy #$ff
a525 c8        getst:    iny
a526 20 fc a2            jsr basin           ;String einlesen
a529 99 3c 03            sta mem,y           ;und zwischenspeichern
a52c a5 90               lda status          ;Diskstatus
a52e f0 f5               beq getst
a530 98                  tya                 ;Laenge merken
a531 48                  pha
a532 20 4e 9b            jsr reservstr
a535 68                  pla                 ;aus Puffer in String schreiben
a536 a8                  tay
a537 88                  dey
a538 b9 3c 03  putst:    lda mem,y
a53b 91 62               sta ($62),y
a53d 88                  dey
a53e 10 f8               bpl putst           ;String als Ergebnis liefern
a540 20 56 9b            jsr string
a543 20 31 a3            jsr readfileclo     ;Datei schliessen
a546 4c 73 00            jmp chrget
```

Listing 4/6.4.2.22

```
Beispiel:
PRINT ST$
00, OK, 00, 00
READY.
```

4/6.4.2.23

COLLECT

Der Befehl COLLECT dient dem 'aufräumen' der Diskette. Er ersetzt die Befehlsfolge

```
OPEN1,8,15,"V":CLOSE1
```

Collect schreibt zunächst den Befehl in den Puffer ab <mem>. Dann wird der Accu mit der Befehlslänge geladen. Ab <sendmem> wird nun der Befehlskanal der Floppy geöffnet, und die Zeichenkette ab <mem> als Befehl gesendet. Wir werden <sendmem> noch für andere Befehle benutzen. Danach wird der Befehlskanal wieder geschlossen.

```
                ;---------------------------------------
                ;--- XBASIC-Befehl  COLLECT          ---
                ;---------------------------------------
a549 a9 56      BBcollect: lda #'v          ;OPEN dateinr,8,15,"v"
a54b 8d 3c 03              sta mem          ;Dateiname
a54e a9 01                 lda #1           ;1 Byte lang
a550 a2 3c      sendmem:   ldx #<mem
a552 a0 03                 ldy #>mem
a554 20 06 a3              jsr filename
a557 ad ab a4              lda dateinr
a55a a2 08                 ldx #8
a55c a0 0f                 ldy #15
a55e 86 02                 stx xmem         ;Setze Fileparameter
a560 a2 1a                 ldx #26
a562 20 b6 9a              jsr basicrom
a565 a2 1e                 ldx #30          ;open
a567 20 b6 9a              jsr basicrom
a56a ad ab a4              lda dateinr      ;close
a56d a2 20                 ldx #32
a56f 4c b6 9a              jmp basicrom
```

Listing 4/6.4.2.23

4/6.4.2.24

INITIALISE

INITIALISE initialisiert die Diskettenstation d.h., die Block-Belegungstabelle der Diskette wird in den Speicher der Diskettenstation gelesen. INITIALISE entspricht der Befehlsfolge:

```
OPEN1,8,15,"I":CLOSE1
```

Die Routine schreibt lediglich den Befehl in den Puffer ab <mem> und lädt den Accu mit der Befehlslänge. Alles weitere übernimmt, wie beschrieben, die Routinen <sendmem> des Befehls COLLECT.

```
                ;-------------------------------------
                ;--- XBASIC-Befehl  INITIALISE     ---
                ;-------------------------------------
a572 a9 49      BBinitial: lda #'i           ;'i' als Datinamen
a574 8d 3c 03              sta mem
a577 a9 01                 lda #1
a579 d0 d5                 bne sendmem       ;weiter bei COLLECT
```

Listing 4/6.4.2.24

4/6.4.2.25

HEADER FORM$

Zum Formatieren von Disketten kann der Befehl HEADER benutzt werden. Er bietet alle Möglichkeiten, welche Sie vom NEW-Befehl der Diskettenstation gewohnt sind. HEADER FORM$ ersetzt nämlich nur die Befehlsfolge

```
OPEN1,8,15,"N:"+FORM$:CLOSE1
```

Beispiel:

HEADER "TEST,ID"	formatiert eine neue Diskette mit Namen "TEST" und der Kennung "ID"
HEADER "TEST"	löscht Inhaltsverzeichnis und BAM und gibt der Diskette den Namen "TEST"

Die Routine des Befehl hat nur die Aufgabe den Akkumulator mit dem Befehlsnamen ('N') zu laden. Die Ausführung wird dann bei dem Befehl SCRATCH vorgenommen, da sich dieser nur im Befehlsnamen von HEADER unterscheidet.

```
             ;--------------------------------------
             ;--- XBASIC-Befehl  HEADER FORM$    ---
             ;--------------------------------------
a57b a9 4e   BBheader:  lda #'n           ;Befehl 'n'
a57d 2c                 bit
```

Listing 4/6.4.2.25

4/6.4.2.26

SCRATCH FILES$

SCRATCH entspricht dem SCRATCH-Befehl der Diskettenstation. Mit ihm können Dateien auf der Diskette gelöscht werden. 'SRATCH FILES$' ersetzt also die Befehlsfolge:

```
OPEN1,8,15,"S:"+FILES$:CLOSE1
```

Beispiel:

SCRATCH "TEST"	löscht die Datei "TEST"
SCRATCH "A*"	löscht alle mit 'A' beginnenden Dateien

Die Routine schreibt zunächst den Befehlsnamen in den Puffer ab <mem>. Dies kann der Name 'S:' sein, oder aber der Name 'N:', falls der Einsprung in die Routine vom Befehl HEADER erfolgt. Dann wird der nachfolgende String geholt, und in dem Puffer an den Befehlsnamen angehängt. Die Routine <sendmem> des COLLECT-Befehl übernimmt wieder die Übermittlung des so gewonnenen Befehls an die Diskettenstation.

```
                ;---------------------------------------
                ;--- XBASIC-Befehl  SCRATCH FILES$   ---
                ;---------------------------------------
a57e a9 53      BBscratch: lda #'s              ;Befehl 's'
a580 8d 3c 03              sta mem
a583 a9 3a                 lda #':
a585 8d 3d 03              sta mem+1
a588 20 0b 9b              jsr frmevl           ;String FILES$ holen
a58b a2 04                 ldx #4
a58d 20 b6 9a              jsr basicrom
a590 98                    tya                  ;Laenge in Accu
a591 48                    pha
a592 88                    dey
a593 b1 22      addstring: lda ($22),y          ;String in mem+2 uebertragen
a595 99 3e 03              sta mem+2,y
a598 88                    dey
a599 10 f8                 bpl addstring
a59b 68                    pla                  ;Laenge des Strings
a59c 18                    clc
a59d 69 02                 adc #2               ;plus 2 ist Laenge des Befehls
a59f d0 af                 bne sendmem          ;weiter bei COLLECT
```

Listing 4/6.4.2.26

4/6.4.2.27

RENAME A$ TO N$

Das Umbenennen von Dateien ermöglicht der Befehl RENAME. Er hat folgenden Aufbau:

RENAME alter Name TO neuer Name

Beispiel:
RENAME "test" TO "probe" benennt die Datei "test" in "probe" um.

Die entsprechende Befehlsfolge im Standard-Basic des C 64 lautet:

```
OPEN1,8,15,"R:"+N$+"="+A$:CLOSE1
```

in unserem Beispiel also:

```
OPEN1,8,15,"R:PROBE=TEST":CLOSE1
```

Auch die Routine des RENAME-Befehls setzt den Befehlsstring im Puffer <mem> zusammen. Sie beginnt mit dem Befehlsnamen 'R:', welcher ab <mem> gespeichert wird. Dann wird der erste String eingelesen, und ab <mem>+40 zwischengespeichert. Dies ist nötig, da wir die Reihenfolge der Dateinamen für den RENAME-Befehl der Diskettenstation umdrehen müssen.

Als nächstes wird überprüft, ob die Zeichenketten durch den TO-Befehl abgetrennt sind. Ist dies der Fall, kann der zweite String eingelesen, und an den Befehlsnamen angehängt werden. Dann muß nur noch das Zeichen '=', sowie der zwischengespeicherte, erste Name angehängt werden. Die Länge des erzeugten Befehls wird in den Akkumulator geladen, und das ganze wieder der Routine <sendmem> übergeben.

```
                ;--------------------------------------
                ;--- XBASIC-Befehl  RENAME A$ TO N$  ---
                ;--------------------------------------
a5a1 a9 52      BBrename:   lda #'r          ;Befehl 'r'
a5a3 8d 3c 03               sta mem
a5a6 a9 3a                  lda #':
a5a8 8d 3d 03               sta mem+1
a5ab 20 0b 9b               jsr frmevl       ;String A$ holen
a5ae a2 04                  ldx #4
a5b0 20 b6 9a               jsr basicrom
a5b3 84 fd                  sty azg2         ;Laenge in azg2
a5b5 a9 00                  lda #0           ;ab mem+42 String zwischenspeichern
a5b7 99 66 03               sta mem+42,y     ;und mit Null abschliessen
a5ba 88                     dey
a5bb b1 22      savestring:lda ($22),y
a5bd 99 66 03               sta mem+42,y
a5c0 88                     dey
a5c1 10 f8                  bpl savestring
a5c3 a9 a4                  lda #164         ;auf 'TO' testen
a5c5 a2 00                  ldx #0
a5c7 20 b6 9a               jsr basicrom
a5ca 20 0b 9b               jsr frmevl       ;String N$ holen
a5cd a2 04                  ldx #4
a5cf 20 b6 9a               jsr basicrom
a5d2 84 fe                  sty azg2+1       ;Laenge in azg2+1
a5d4 88                     dey              ;String an Befehl 'r:' anhaengen
a5d5 b1 22      addstring1:lda ($22),y
a5d7 99 3e 03               sta mem+2,y
a5da 88                     dey
a5db 10 f8                  bpl addstring1

a5dd a6 fe                  ldx azg2+1       ;'=' anhaengen
a5df a9 3d                  lda #'=
a5e1 9d 3e 03               sta mem+2,x

a5e4 a0 ff                  ldy #$ff         ;ersten String anhaengen
a5e6 e8         addstring2:inx
a5e7 c8                     iny
a5e8 b9 66 03               lda mem+42,y
a5eb 9d 3e 03               sta mem+2,x
a5ee d0 f6                  bne addstring2

a5f0 18                     clc              ;Befehlslaenge ist Stringlaege 1 & 2
a5f1 a9 03                  lda #3           ;plus 3 Byte fuer 'r:' bzw. '='
a5f3 65 fd                  adc azg2
a5f5 65 fe                  adc azg2+1
a5f7 4c 50 a5               jmp sendmem      ;weiter bei COLLECT
```

Listing 4/4.6.2.27

4/6.4.2.28

PAUSE TIME

Der Pause-Befehl wurde aus der Basic-Erweiterung des C 128 übernommen, und an unsere Erweiterung angepaßt. Er hält den Computer für eine Zeit zwischen 1/100 Sekunde und 655.35 Sekunden (ca. 10 Minuten und 55 Sekunden) an. Ein Ausstieg aus der Funktion ist mit der STOP-Taste möglich. Eine genaue Beschreibung des Befehls finden Sie in den Kapiteln 4/6.5.3 und 4/6.5.3.1.

Beispiel:
PAUSE 10.5 wartet 10.5 Sekunden

```
             ;---------------------------------------
             ;--- XBASIC-Befehl  PAUSE TIME        ---
             ;---------------------------------------
a5fa 20 0b 9b BBpause:   jsr frmevl         ;Ausdruck holen
a5fd 24 0d               bit $d             ;Typ testen
a5ff 10 03               bpl typok2
a601 4c 11 9f            jmp typerr         ;Typfehler
a604 a9 10    typok2:    lda #<fk100        ;mit 100 multiplizieren
a606 a0 9d               ldy #>fk100
a608 a2 32               ldx #50
a60a 20 b6 9a            jsr basicrom
a60d 20 2b 9b            jsr adresse        ;und in INTEGER wandeln
a610 a9 7a               lda #<9850         ;Teilrate A setzen
a612 8d 04 dd            sta cia2+4
a615 a9 26               lda #>9850
a617 8d 05 dd            sta cia2+5
a61a a5 14               lda $14            ;Teilrate B setzen
a61c 8d 06 dd            sta cia2+6
a61f a5 15               lda $15
a621 8d 07 dd            sta cia2+7
a624 ad 0e dd            lda cia2+14        ;Kontrollreg. A
a627 29 d1               and #%11010001
a629 09 11               ora #%00010001
```

Listing 4/6.4.2.28 (Teil1)

```
a62b 8d 0e dd          sta cia2+14
a62e ad 0f dd          lda cia2+15        ;Kontrollreg. B
a631 29 d1             and #%11010001
a633 09 59             ora #%01011001
a635 8d 0f dd          sta cia2+15

a638 ad 0d dd pauloop: lda cia2+13        ;Interrupt-Flag-Reg.
a63b 29 02             and #%10
a63d d0 07             bne pauend
a63f a2 30             ldx #48            ;STOP-Taste abfragen
a641 20 b6 9a          jsr basicrom
a644 d0 f2             bne pauloop        ;warten
a646 60       pauend:  rts
```

Listing 4/6.4.2.28 (Teil2)

4/6.4.2.29

A$=TIME X

Die Funktion TIME wurde ebenfalls aus der C 128 Erweiterung übernommen. Sie dient dem Auslesen einer Echtzeituhr. Von dieser Uhr ist jeweils eine in CIA1 und eine in CIA2 vorhanden. TIME1 liest die Uhr aus CIA1, und TIME2 die Uhr aus CIA2. Eine genaue Beschreibung des Befehls finden Sie in den Kapiteln 4/6.5.3 und 4/6.5.3.3.

Beispiel:
PRINT TIME 1 12:34:33:8

```

                   ;--------------------------------------
                   ;--- XBASIC-Befehl   A$=TIME X       ---
                   ;--------------------------------------
a647 20 73 00      BBtime:    jsr chrget        ;Uhrnummer holen
a64a 20 33 9b                 jsr getbyte
a64d ca                       dex               ;minus1
a64e e0 02                    cpx #2            ;Uhrnummer <=1 ?
a650 b0 77                    bcs timeerr       ;nein
a652 8a                       txa               ;Uhrnummer+$dc
a653 18                       clc
a654 69 dc                    adc #$dc
a656 85 fc                    sta azg+1         ;azg auf CIA-Basis
a658 a9 00                    lda #0
a65a 85 fb                    sta azg
a65c 85 fd                    sta azg2          ;Zaehler auf Null
a65e a9 0a                    lda #10           ;10 Zeichen
a660 20 4e 9b                 jsr reservstr     ;String einrichten
a663 a0 0b                    ldy #11
a665 b1 fb                    lda (azg),y       ;Stunden
a667 f8                       sed               ;BCD-Arithmetik
a668 08                       php               ;Flag retten
a669 29 7f                    and #%01111111    ;ohne PM-Flag
a66b c9 12                    cmp #$12          ;12 Uhr
a66d d0 02                    bne time3
a66f a9 00                    lda #0            ;12 Uhr AM = 0 Uhr
a671 28            time3:     plp               ;Flag holen
a672 10 03                    bpl time4         ;AM
a674 18                       clc
a675 69 12                    adc #$12          ;+ 12 Stunden
a677 d8            time4:     cld               ;BCD aus
a678 20 a2 a6                 jsr timeud        ;-> String
a67b a0 0a                    ldy #10
a67d b1 fb                    lda (azg),y       ;Minuten
a67f 20 a2 a6                 jsr timeud        ;-> String
a682 a0 09                    ldy #9
a684 b1 fb                    lda (azg),y       ;Sekunden
a686 20 a2 a6                 jsr timeud        ;-> String
a689 a0 08                    ldy #8
a68b b1 fb                    lda (azg),y       ;1/10 Sekunden
a68d 20 97 a6                 jsr timeu2        ;-> String
a690 4c 56 9b                 jmp string

a693 4a            timeu1:    lsr               ;High-Nibble -> String
a694 4a                       lsr               ;4 Bit nach rechts
a695 4a                       lsr
a696 4a                       lsr
a697 29 0f         timeu2:    and #$f           ;Low-Nibble -> String
a699 09 30                    ora #'0
a69b a4 fd         timeu3:    ldy azg2          ;Accu -> String
a69d 91 62                    sta ($62),y
a69f e6 fd                    inc azg2
a6a1 60                       rts

a6a2 48            timeud:    pha               ;Byte eintragen
a6a3 20 93 a6                 jsr timeu1        ;High Nibble eintragen
a6a6 68                       pla
a6a7 20 97 a6                 jsr timeu2        ;Low Nibble eintragen
a6aa a9 3a                    lda #':           ;':' eintragen
a6ac 20 9b a6                 jsr timeu3
a6af 60                       rts

```

Listing 4/6.4.2.29

4/4.4.2.30

SETTIME NR,ZEIT$

Mit dem Befehl SETTIME kann eine der Uhren oder der Alarmzeiten in den CIA-Bausteinen gestellt werden. Auch der SETTIME-Befehl stammt aus der C 128-Erweiterung, und kann in den Kapiteln 4/6.5.3 und 4/6.5.3.2 nachgelesen werden.

NR ist eine Nummer, welche die angesprochene Uhr identifiziert. Es gilt folgende Zuordnung:

NR=1	Uhr in CIA1
NR=2	Uhr in CIA2
NR=3	Alarmzeit in CIA1
NR=4	Alarmzeit in CIA2

Zeit$ ist eine Zeichenreihe aus 10 Zeichen im Format "HH:MM:SS:Z", wobei HH die Stunden, MM die Minuten, SS die Sekunden und Z die Zehntelsekunden beschreibt.

Beispiel:
Uhr in CIA2 auf 13 Uhr 34 Minuten und 5 Sekunden setzen:
SETTIME 2,"13:34:05:0"

```
                ;-------------------------------------
                ;--- XBASIC-Befehl  SETTIME  NR,ZEIT ---
                ;-------------------------------------
a6b0 20 33 9b   BBsettime: jsr getbyte      ;Uhrnummer holen
a6b3 ca                    dex              ;1 abziehen
a6b4 86 fd                 stx azg2         ;Clocknr. (0,1,2,3)
a6b6 e0 04                 cpx #4           ;Nr. <=4 ?
a6b8 b0 0f                 bcs timeerr      ;ein, ERROR
a6ba 20 23 9b              jsr chkkom       ;Komma testen
a6bd 20 0b 9b              jsr frmevl       ;Zeit-String holen
a6c0 a2 04                 ldx #4
a6c2 20 b6 9a              jsr basicrom
a6c5 c0 0a                 cpy #10          ;10 Zeichen ?
a6c7 f0 03                 beq settime1     ;OK
a6c9 4c 11 a1   timeerr:   jmp zugross      ;Fehler
a6cc a9 00      settime1:  lda #0
a6ce 85 fe                 sta azg2+1       ;Zaehler=0
a6d0 46 fd                 lsr azg2         ;Clocknr: 0-BIT in Carry
a6d2 a9 00                 lda #0           ;Zeiger auf CIA
a6d4 85 fb                 sta azg
a6d6 69 dc                 adc #$dc         ;High-Byte von CIA
a6d8 85 fc                 sta azg+1
a6da a0 0e                 ldy #14          ;Kontrollreg. A
a6dc b1 fb                 lda (azg),y
a6de 09 80                 ora #%10000000   ;50Hz-Flag setzen
a6e0 91 fb                 sta (azg),y
a6e2 c8                    iny
a6e3 a5 fd                 lda azg2         ;Clocknr /2
a6e5 4a                    lsr              ;0-Bit in Carry
a6e6 08                    php              ;Carry sichern
a6e7 b1 fb                 lda (azg),y      ;Kontrollreg. B
a6e9 2a                    rol
a6ea 28                    plp              ;Carry von oben
a6eb 6a                    ror
a6ec 91 fb                 sta (azg),y      ;Alarmzeit / Uhrzeit
a6ee 20 17 a7              jsr ausw2        ;2 Ziffern ausw.
a6f1 f8                    sed              ;BCD-Arithmetik
a6f2 c9 12                 cmp #$12         ;12 Uhr
a6f4 f0 06                 beq sett2        ;PM-Flag autom.
a6f6 90 04                 bcc sett2        ;vor 12 Uhr
a6f8 e9 12                 sbc #$12         ;12 abziehen
a6fa 09 80      settpm:    ora #%10000000   ;PM-Flag
a6fc d8         sett2:     cld              ;BCD aus
a6fd a0 0b                 ldy #11
a6ff 91 fb                 sta (azg),y      ;Stunden setzen
a701 20 17 a7              jsr ausw2        ;2 Ziffern ausw.
a704 a0 0a                 ldy #10
a706 91 fb                 sta (azg),y      ;Minuten setzen
a708 20 17 a7              jsr ausw2        ;2 Ziffern ausw.
a70b a0 09                 ldy #9
a70d 91 fb                 sta (azg),y      ;Sekunden setzen
a70f b1 22                 lda ($22),y      ;1/10 Sekunden
a711 29 0f                 and #$f
a713 88                    dey              ;setzen
a714 91 fb                 sta (azg),y
```

Listing 4/6.4.2.30 (Teil1)

```
a716 60                    rts
a717 a4 fe        ausw2:   ldy azg2+1      ;2 Ziffern & ':'
a719 b1 22                 lda ($22),y     ;naechstes Zeichen
a71b 0a                    asl             ;4 Bit nach links
a71c 0a                    asl
a71d 0a                    asl
a71e 0a                    asl
a71f 8d 3c 03              sta mem         ;merken
a722 c8                    iny
a723 b1 22                 lda ($22),y     ;naechstes Zeichen
a725 29 0f                 and #$f         ;4 Bit ausblenden
a727 0d 3c 03              ora mem         ;mit 1. Ziffer odern
a72a 8d 3c 03              sta mem         ;merken
a72d c8                    iny
a72e b1 22                 lda ($22),y     ;naechstes Zeichen
a730 c9 3a                 cmp #$3a        ;':'
a732 f0 05                 beq ausw21      ;ok
a734 68                    pla             ;Stack bereinigen
a735 68                    pla
a736 4c 11 a1              jmp zugross     ;Fehler
a739 ad 3c 03     ausw21:  lda mem         ;Ergebnis in Accu
a73c c8                    iny
a73d 84 fe                 sty azg2+1      ;Zaehler merken
a73f 60                    rts
```

Listing 4/4.6.2.30 (Teil2)

4/6.4.2.31

SETEOL X

Der Befehl SETEOL X legt das EOL-Zeichen für den Befehl LINEINPUT# im Speicher ab. Auf die Bedeutung dieses Zeichens wird bei dem Befehl LINEINPUT# eingegangen. Als Standardzeichen ist das Zeichen 13 (= <RETURN>) abgelegt.

```
                ;-------------------------------------
                ;--- XBASIC-Befehl  SETEOL X       ---
                ;-------------------------------------
                eolst:       .eq 13           ;Standard End of Line

a740 20 33 9b   BBseteol:    jsr getbyte      ;EOL holen
a743 8e 47 a7                stx eolbyte
a746 60                      rts

a747 0d         eolbyte:     .by eolst        ;EOL-Byte
```

Listing 4/6.4.2.31

4/6.4.2.32

LINEINPUT A$

LINEINPUT liest maximal 80 Zeichen von der Tastatur ein. Dabei wird jedes Zeichen akzeptiert, also auch die Zeichen ',' und ':', welche von dem normalen INPUT-Befehl zurückgewiesen werden. Die Zeichen werden in den Puffer <inpuffer> abgelegt. Dieser Puffer befindet sich am Ende der Basicerweiterung, also unter den Basic-ROM.

Das Einlesen wird beendet, wenn die RETURN-Taste gedrückt wurde, oder die maximale Zeichenzahl überschritten wurde.

Wurde die Eingabe mit <RETURN> abgeschlossen, wird ein String mit der erforderlichen Länge angelegt, und die Zeichen in diesen String übertragen. Die Adresse des Strings wird der angegebenen Variablen übergeben.

```
               ;---------------------------------------
               ;--- XBASIC-Befehl  LINEINPUT A$      ---
               ;---------------------------------------
a748 a9 50     BBlineinput:lda #80          ;max. 79 Zeichen
a74a 8d 3c 03              sta mem
a74d a9 0d                 lda #13          ;EOL ist <RETURN>
a74f 8d 3d 03              sta mem+1
a752 a2 14     dolineinput:ldx #20          ;Variable suchen
a754 20 b6 9a              jsr basicrom
a757 24 0d                 bit $d           ;Typ testen
a759 30 03                 bmi typok3
a75b 4c 11 9f              jmp typerr       ;Typfehler
a75e a0 00     typok3:     ldy #0           ;Zeichenaehler
a760 20 fc a2  linein:     jsr basin        ;ein Zeichen lesen
a763 99 bd a7              sta inpuffer,y   ;und merken
a766 cd 3d 03              cmp mem+1        ;EOL ?
a769 f0 0b                 beq eolfound     ;ja
a76b c8                    iny              ;Zaehler+1
a76c cc 3c 03              cpy mem          ;max. Anzahl ueberschritten ?
a76f d0 ef                 bne linein       ;nein
a771 a2 17                 ldx #23          ;'String too long'
a773 4c 06 9b              jmp fehler

a776 98        eolfound:   tya              ;String anlegen
a777 85 fb                 sta azg
a779 20 4e 9b              jsr reservstr
a77c a4 fb                 ldy azg
a77e 88                    dey
a77f b9 bd a7  setstr:     lda inpuffer,y   ;Zeichen in String uebertragen
a782 91 62                 sta ($62),y
a784 88                    dey
a785 10 f8                 bpl setstr       ;bis alle Zeichen uebertragen
a787 c8                    iny              ;Y=0
a788 a5 fb                 lda azg          ;Laenge in Variable uebertragen
a78a 91 47                 sta ($47),y
a78c c8                    iny
a78d a5 62                 lda $62
a78f 91 47                 sta ($47),y      ;Adresse in Variable uebertragen
a791 c8                    iny
a792 a5 63                 lda $63
a794 91 47                 sta ($47),y
a796 60                    rts
```

Listing 4/6.4.2.32

4/6.4.2.33

LINEINPUT# D,A$

Der LINEINPUT#-Befehl liest eine Zeichenkette von einem beliebigen Gerät. Dabei kann das Ende der Zeichenketten durch ein beliebiges Zeichen markiert sein. Im allgemeinen ist dies das Zeichen 13 (= <RETURN>). Werden die Zeichenketten durch ein anderes Zeichen, z.B. Null, abgeschlossen, so wird der LINEINPUT#-Befehl mit Hilfe des Befehls SETEOL angepaßt. In unserem Beispiel müßte man also 'SETEOL 0' eingeben.

Damit die Zeichenketten in String's passen, dürfen sie die maximale Länge von 255 Zeichen nicht überschreiten.

Die Routine des LINEINPUT#-Befehls leitet die Eingabe auf die angegebene Datei um, und ruft dann den LINEINPUT-Befehl auf. Hat dieser seine Aufgabe beendet, wird die Eingabe wieder auf 'Normal' geschaltet.

```
               ;-----------------------------------------
               ;--- XBASIC-Befehl  LINEINPUT# D,A$   ---
               ;-----------------------------------------
a797 a9 00     BBlineindev:lda #0              ;max. 256 Zeichen
a799 8d 3c 03              sta mem
a79c ad 47 a7              lda eolbyte         ;EOL ist EOLBYTE
a79f 8d 3d 03              sta mem+1
a7a2 20 33 9b              jsr getbyte         ;log. Dateinr. holen
a7a5 86 02                 stx xmem
a7a7 a2 22                 ldx #34             ;Eingabe aus Datei
a7a9 20 b6 9a              jsr basicrom
a7ac b0 0b                 bcs ckerr           ;Fehler
a7ae 20 23 9b              jsr chkkom          ;auf Komma testen
a7b1 20 52 a7              jsr dolineinput     ;LINEINPUT-Routine
a7b4 a2 26                 ldx #38             ;Eingabe normal
a7b6 4c b6 9a              jmp basicrom

a7b9 aa        ckerr:      tax                 ;Fehler melden
a7ba 4c 06 9b              jmp fehler

               inpuffer:
```

Listing 4/6.4.2.33

4/6.4.3.3

Basic-Loader der Version 1.2

Auch für die XBASIC-Version 1.2 wird natürlich wieder ein Basic-Loader angegeben. Dennoch haben Sie es hier etwas schwerer. Durch die Änderung der Adreßlage der Basic-Erweiterung stimmt von den Basic-Loadern der vorangegangenen Versionen leider nichts mehr, was es nötig macht, den Basic-Loader vollständig neu einzugeben. Leider ließ sich dies nicht vermeiden. Dennoch lohnt sich die Arbeit, da man mit den Grafikbefehlen viele Anwendungen erschließt, die sonst nur schwer zugänglich sind.

Wenn Sie sich bisher nicht entschließen konnten, den Quelltext einzugeben, ist jetzt noch einmal Gelegenheit dazu, denn dann trifft Sie das Ändern der Adreßlagen nie mehr. Sie brauchen dann nur neu zu assemblieren, was Sie für die zusätzlichen Befehle ohnehin machen müßten.

```
1 REM XBASIC 1.2 DATA LOADER                  >984
2 :                                           >60
3 Z=2049:PRINT"<CLR><N>ZEILE:",Z,:OPEN1,8,15,"I0":OPEN2,8,2,"XBASIC 1.2,P,W" >3291
4 PRINT#2,CHR$(1);CHR$(8);                    >831
5 READ Z$:IFZ$="--"THENCLOSE2:CLOSE1:END >1007
6 SU=0:FOR I=1 TO 63 STEP 2                   >928
7 W$=MID$(Z$,I,2):IFW$="--"THENAB=1:I=63:GOTO9                >1988
8 GOSUB20:SU=SU+W:PRINT#2,CHR$(W);            >1297
9 NEXT                                        >11
10 W$=MID$(Z$,LEN(Z$)-3,2):GOSUB20:PS=W*256                   >2050
11 W$=RIGHT$(Z$,2):GOSUB20:PS=W+PS            >1378
12 IFSU<>PS THEN PRINT:PRINT"FEHLER IN ZEILE: "Z              >1555
13 IFAB>0THENCLOSE2:CLOSE1:END                >735
14 Z=Z+32:PRINTZ,:GOTO5                       >944
20 W=(ASC(W$)-48+7*(ASC(W$)>57))*16           >1727
21 W$=RIGHT$(W$,1)                            >589
22 W=W+ASC(W$)-48+7*(W$>"9")                  >1619
23 RETURN                                     >37
24 :                                          >82
2049 DATA0B08C3079E32303631000000A932A20885FB86FCA900A29885FD86FEA224A0000D1A >4295
2081 DATAB1FB91FDC8D0F9E6FCE6FECAD0F24C0098A937850120769820179BA91DA09F201289 >3962
2113 DATA2DE4A2FB9A4C86E3A9368D1BD0AD00DD29FC8D00DDA9CC8D8802A900A29885371095 >4222
2145 DATA8638A9A9A2988D18038E1903A929A29A8D04038E0503A9ACA29E8D06038E07030B97 >4777
2177 DATAA971A29F8D08038E0903A945A29F8D0A038E0B03A999A2988D02038E0303A9000B3D >4911
2209 DATA8DF9038DF8036078A50148A210A9D0A00085FC84FBA9328501B1FB91FBC8D0F911CC >3930
2241 DATAE6FCCAD0F46B85015B60A50148A93685012008AA6885014C83A4488A489848A90E3F >4659
2273 DATA7F8D0DDDAC0DDD301D20BCF620E1FFD015A2FB9A2015FD20179B20A3FD2018E50FA5 >4260
2305 DATA2017986C02A04C72FE8D889920E7980848A9368501682B60BD8A9948BD899480DDA >4862
2337 DATAA9378501AD8899A60260E601B1FBC601608622E6014C45A457524F4E47205354D0E >4552
```

Listing 4/6.4.3.3 (Teil 1)

```
2369 DATA52494E474C45CE424144204348415220494E20535452494EC7E6014C37A4E6010AB7 >4871
2401 DATA209EADC60160E60120F7AEC60160E60120FAAEC60160E60120FDAEC60160E6010EFB >3905
2433 DATA20F7B7C60160E601209EB7C60160E601208AAD20F7B7C60160E60120EBB7C6010F1B >3993
2465 DATA60E601207DB4C60160E60120CAB4C60160E60120F1AEC6016000FEAE25B581B70EF6 >3924
2497 DATAF6B848BC3AABE3FF6BE5A5B632A58AB0DCBD7BA5B9FFBCFFBFFFC2FFC5FFC8FF176B >3297
2529 DATACBFFCEFFD1FF289A6AA9E0FF27BA0BBC2ABA6AE263E2A9B1FFFFFFFFFFFFFFFF178A >3626
2561 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >2854
2593 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >2886
2625 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC601208FA4E61CE6 >3082
2657 DATA01D051A0FFA9BA85FBA99A85FCA90185FDCAC8E8B1FBF007DD0002D01CF0F3A41403 >3492
2689 DATA02A9D2990002C8A5FD990002C8BD0002990002F04AE8C8D0F4A60288E6FDC8B11079 >3818
2721 DATAFBD0FB389865FB85FBA900A865FC85FCB1FBD0C0E88602BD0002F01BC983F0171372 >3884
2753 DATAC98FF013C922D09BE8BD0002F009C922D0F6E88602D08CA40BA90099FD01609810B5 >4032
2785 DATAA602E8186905850BD0CA10B828004154280048455824004445430049 4E4B45590942 >5053
2817 DATA2400494E535452280053545249 4E4724280053504143452428004452454 54B000722 >5290
2849 DATA5245454B004445454B00535424005449 4D45005445535452280054455354280075C >5347
2881 DATA434C53004F4C4443484152004336340058424153494300494E5354009643484107E6 >5303
2913 DATA52004F4C440044524F4B4500524F4B4500444F4B45004155A40053574150004407B3 >5245
2945 DATA495200519300434F4C4C45435400484541444552004449520052454E414D450007C5 >5364
2977 DATA5343524154434800494E495449414C4953450053455444C152005041555345000B B4 >5274
3009 DATA53455454494D450053554454F4C004C494E45585004C494E45584004452415 74D08BB >5539
3041 DATA4F4445004280444552004 94E4B0050415045520050454E00475241469100475208 2C >5439
3073 DATA41464F4646004752414 69C004D554C544991004D4F564552004D4F564500504C0896 >5443
3105 DATA4F545200504C4F54004C494E4552004C494E450046524 14D4500424F5800434907B6 >5336
3137 DATA52434C4500414E474C4500434F4C424C4F434B00424C4F434B0000FFFFFFFFFF0B3C >5129
3169 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >2950
3201 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >2982
3233 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >3014
3265 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >3046
3297 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >3078
3329 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >2854
3361 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >2886
3393 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >2918
3425 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >2950
3457 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >2982
3489 DATAFFFFFFFFFFFF874800000007B0EFA351231A11FA2BCA164A1F5A1D3A26CA25011F0 >3788
3521 DATAA237A557A574A54EA681A738AB41AB43E57598E1FCFF97E59F53A0CC9F10A5D31290 >4008
3553 DATAA4E7A478A494A5E6A55EA383A6B5A60000DBA6B8A6ACA6DEA534A7EAA77AA8B21358 >3361
3585 DATAA8D1A818A9F7A80DA902A92CA973A907AA4AA9C0A9F4AACCAA77AB7DAB3BAC531178 >3405
3617 DATAAC92ADF3AD3DAF8CB141B3E9AEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1B2C >3181
3649 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >2918
3681 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >2950
3713 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >2982
3745 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1FE0 >3014
3777 DATAFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1064481D9F >3390
3809 DATAA9BA85FBA99A85FC68C9D2D059240F3052C8B15FAA8449A000CAF013C8B1FBD01288 >3931
3841 DATAFB3898A00065FB85FB90EEE6FCD0EAB1FB300BF0342047ABC8D0F4840238E97F1331 >3820
3873 DATAAAA0FFCAF008C8B99EA010FA30F5C8B99EA030052047ABD0F5297F2047ABA4021124 >3674
3905 DATAC8D0CC4CF3A64C1CA7A4494CF6A693D8C2C1D3C9C320D6455253494F4E20312E10BF >3845
3937 DATA320D28432920CD2EC652494553452031393837 0D0D00A900850D20730008C9D208B0 >4830
3969 DATAF004284C8DAE28207300CDB99AB0F4C60120669FE601600AAABD7F9D48BD7E9D0F02 >4588
4001 DATA48602073000 8C9D2F008C98BF025284CE7A768C601208D9FE6014CAEA76020730E37 >4497
4033 DATA00CDB99A90F70AAABD7F9D48BD7E9D484C73002820730020 9EAD207900C989F00E5C >4873
4065 DATA05A9A720FFAEA561D0092009A920FBA84CAEA7207900B0AD20A0A84CAEA7A52C0F07 >3833
4097 DATAA001912BA21220D898A52218690285 2DA5236900852E60202D99A20420D898C0889 >4697
4129 DATA3C03A52285FDA52385FE204599A21420D898A50D1047A002B14785FC88B147850E40 >4352
4161 DATAFB204599205599CAE0FFD005A20E4C28998E3D0318 8A6D3C03B025E900A000D10D8D >4563
4193 DATA47B01DAD3D0365FB85FB9002E6FCAC3C03AE3C0388B1FD91FBCAD0F860A2162C10C5 >4038
4225 DATAA20F4C2899205599B6FBA90085FC06FB26FC06FB26FC06FB26FC18A90065FB850F86 >4493
4257 DATAFBA9D065FC85FCA9088D3E03204599205599E002B0128A0A0A0A65FC85FC20450E74 >4684
4289 DATA99205599E00490034C4CA28A2901B5FD8A4A85FEA9008D3D03204599202D99A20CE1 >5007
4321 DATA0420D8988D3C03A0FFA2088602A6FDF0024602C8B122C92EF01438E940905DF00EE2 >4253
4353 DATA5BC904B057A6FDD007C902B04F2CA900A6FDD0076A2E3F034CF3A06A6A2E3F030DBF >4108
```

Listing 4/6.4.3.3 (Teil 2)

```
4385 DATA2A2E3F03C602D0CB984BA5014878A9328501AC3D03AD3F03A6FEF00249FF91FB0DE9 >4233
4417 DATA68850158EE3D0368A8CE3E03D09BC8CC3C03D00160A207A9994C0099A216A9990DCC >4349
4449 DATA4C0099207300202D99203599A90DF005A21020D898A20220D89838A533E531A80C3B >4561
4481 DATAA534E532A200860D85628463A290860238A2084CD898207300208099A20420D80CB5 >4585
4513 DATA98D005A2064CD89898AAA00084148415B122C9309020E930C90A9008E90790160D7A >4558
4545 DATAC910B01220ACA11865148514A51569008515C8CAD0DAA515A4144C57A10614260C21 >4305
4577 DATA15061426150614261506142615602073002080992O4D99A904207099A003A5140778 >5176
4609 DATA290F186930C93A9002690691622OE5A18810EB4C78994615661446156614461 50A6B >5016
4641 DATA661446156614 60A90085CF85CCA20C20D898C900F0F748A90185CCA5CEA4D3910FA9 >3971
4673 DATAD1A901207099A0006891622078994C7300207300205599E019B02286FB2045990C7A >4850
4705 DATA205599E028B01686D3A6F886D6A20E20DB98203599A9002070994C7899A20E4C0E8B >4575
4737 DATA28992073002055998A48207099 68A8F007A92088 9162D0FB2078994C359920730D51 >5077
4769 DATA00202D99A20420D898A52285FDA52385FE84FB204599205599 86FC8AF035A6FB100D >4556
4801 DATAF031A9001865FCB037CAD0F9207099A6FCA00084FCB1FD8402A4FC9162A402E611FB >3844
4833 DATAFCC8C4FBD0EFA000CAD0EAC60D2078994C3599A900207099C60D2078994C35991074 >4437
4865 DATAA2174C2899207300205599 8E3C03204599202D99A20420D898A52285FBA523850BE2 >4721
4897 DATAFC8C3D0320459920 2D99A20420D898A52285FDA52385FE8C3E03203599AC3C030D1C >4743
4929 DATAD005A20E4C2899888C4103A9008D4003AD41038D3F03AC3F03CC3E03B01BB1FD0B97 >4645
4961 DATAAC4003CC3D03F01FD1FBD008EE3F03EE4003D0E2EE4103D0D2AC4003CC3D03F00F80 >3958
4993 DATA06A900A84C53A1A900AC4103C84C53A1A63AEBF005A21B4C2899A52B85AEA52C0D93 >4155
5025 DATA85AF202D99A20420D89898A622A4232041A4207900A208C92CD0062045992055 0BFD >4530
5057 DATA99ADE6A5A0002048A4A0FFC8C058F0762037A4990002C90DD0F1AD0002C930900FC7 >3872
5089 DATA49C93AB045A900990002A9FF857AA901857B207300A22E20D898A22C20D898A00DC7 >4450
5121 DATA00A90891AEC891AEC8A51491AEC8A51591AEC8A200BD000291AEE8C8C900D0F5111E >3516
5153 DATA981865AE85AE9002E6AFA590F09BA90091AEC891AEA8917AC8C8917A206CA4A911F3 >3966
5185 DATA0085C64CCD9FA000B90002C930900AC939B006203CA4C8D0EF2037A4C90DD0F90ECA >3887
5217 DATAA92C203CA44CF9A3A2284CD898A22A4CD8988602A21C4CD8988602A21A20D8980EA1 >4284
5249 DATAA21E20D898B00DADE6A58502A22220D898B0016048206CA468AA4C2899ADE6A50F00 >4151
5281 DATAA22020D898A2264CD898205D99A5148D8DA4A5158D8EA405148D8CA460010A000D1D >4738
5313 DATAAD00024BA21820D898840B68AE8CA4F031C930902DC93AB02918A514486D8DA40D80 >4544
5345 DATAA8A515486D8EA42053A1A21620D898A2FFE8BD01019D7702D0F786C6688515680F80 >4379
5377 DATA8514A2FF60206899A000A5014878A92285018A911468850160205D99A514A6150C19 >4674
5409 DATA85FB86FC204599205D99A9918DFB98A000A51420F998C8A51520F998A9B18DFB118F >4283
5441 DATA9860205D99A514A61585FB86FC204599205D9978A9228501A514A00091FBC8A50EB3 >4418
5473 DATA1591FBA936850158602073002080992O4D9978A001A9348501B114AA8BB114A80C70 >4662
5505 DATAA93685015 88A4C53A1207300208099204D9978A000A9348501B114A8A93685010BA6 >4685
5537 DATA58A9004C53A1207300208099204D99A51485FBA51585FCA00120F998AA8820F90E24 >4580
5569 DATA98A88A4C53A1A21420D898A50DB5FDA50E85FEA54785FBA54885FC204599A2141048 >4394
5601 DATA20DB98A50DC5FDD006A50EC5FEF0034C4CA0A50DD007A50ED006A0042CA0022C0E2B >4044
5633 DATAA001B1FBAAB14791FB8A91478810F3602055998EE6A56050202D99A20420D8980FBB >4275
5665 DATA98A622A4232041A4ADE6A5A208A0002048A42037A42037A42037A42037A420370BFD >3904
5697 DATAA4A82037A4A690D01E2039A6A920203CA42037A4C900F006203CA44C20A6A90D0CB5 >3886
5729 DATA203CA4A590F0D14C6CA42053A1A21620D898A000B90101F006203CA4C8D0F5600EEC >3854
5761 DATAA9002041A4ADE6A5A208A00F2048A4A0FFC82037A4993C03A590F0F5984820700F0F >4011
5793 DATA9968A888B93C0391628810F8207899206CA44C7300A9568D3C03A901A23CA0030CC2 >4699
5825 DATA2041A4ADE6A5A208A00F8602A21A20D898A21E20D898ADE6A5A2204CDB98A9490F62 >4008
5857 DATA8D3C03A901D0D5A94E2CA9538D3C03A93A8D3D03202D99A20420D89898488 8B10CEB >4738
5889 DATA22993E038810F868186902D0AFA9528D3C03A93A8D3D03202D99A20420D898840C0D >4776
5921 DATAFDA900996603 8BB12299660388 10F8A9A4A20020DB98202D99A20420D89884FE0EB2 >4256
5953 DATA88B122993E038810F8A6FEA93D9D3E03A0FFEBC8B966039D3E03D0F61BA903650F6B >4289
5985 DATAFD65FE4C8BA6202D99240D10034C4CA0A976A09DA23220D898204D99A97A8D040DB9 >4634
6017 DATADDA9268D05DDA5148D06DDA5158D07DDAD0EDD29D109118D0EDDAD0FDD29D1090E2F >4310
6049 DATA598D0FDDAD0DDD2902D007A23020D898D0F2602073002055 99CAE002B0778A180E05 >4495
6081 DATA69DC85FCA90085FB85FDA90A207099A00BB1FBF808297FC912D002A9002810030EDD >4274
6113 DATA1B6912D820DDA7A00AB1FB20DDA7A009B1FB20DDA7A008B1FB20D2A74C78994A1096 >3865
6145 DATA4A4A4A290F0930A4FD9162E6FD604B20CEA76820D2A7A93A20D6A760205599CA0EBC >4193
6177 DATA86FDE004B00F204599202D99A20420D898C00AF0034C4CA2A90085FE46FDA9000E4F >4296
6209 DATA85FB69DC85FCA00EB1FB098091FBC8A5FD4A08B1FB2A286A91FB2052A8F8C9121252 >4024
6241 DATAF0069004E91209B0D8A00B91FB2052A8A00A91FB2052A8A00991FBB122290F880E4F >4318
6273 DATA91FB60A4FEB1220A0A0A0A8D3C03C8B122290F0D3C038D3C03C88122C93AF0050BD3 >4496
6305 DATA68684C4CA2AD3C03C884FE602055998E82A8600DA9508D3C03A90D8D3D03A2140CD1 >4785
6337 DATA20D898240D30034C4CA0A0002037A4990005CD3D03F00BC8CC3C03D0EFA2174C0C04 >4487
6369 DATA28999885FB207099A4FB88B900059162 8810FBC8A5FB9147C8A5629147C8A5631121 >4488
```

Listing 4/6.4.3.3 (Teil 3)

```
6401 DATA914760A9008D3C03AD82A88D3D032055998602A22220DB98B00B204599208DA80C49 >4742
6433 DATAA2264CD898AA4C2899205599E010B0298E20D060205599E010B01E8E21D060200DBB >4469
6465 DATA5599E010B0138E860260205599E002B0088A186A6A8DF90360A20E4C289920550C50 >4906
6497 DATA99E004B0F48AA2009D43A90A0AEBE004D0F6600101041040FCF3CF3FA0008CFA0F55 >4135
6529 DATA038CFB038CFC03A9E085FC84FBA9F085FE84FDA2109891FB91FDC8D0F9E6FCE615FB >3775
6561 DATAFECAD0F260A9282027AA8D18D0A93B8D11D02CF8031003205DAA205599E010B00E82 >4300
6593 DATA2EB6FC204599205599E0108AB02106FC06FC06FC06FC05FCA0C884FCA204A0000F3E >4376
6625 DATA84FB91FBC8D0FBE6FCCAD0F64C40AA4C28A92CF8033003205DAA2055992027AA10E3 >4142
6657 DATAE010B0EB8E21D02045992089A9204599205599E010B0D8A0008A9900DB9900D90EEB >4219
6689 DATA9900DA9900DBC8D0F1A9288D18D0A93B8D11D0A9188D16D0602CF8031003205D0E53 >4283
6721 DATAAAA9008DF803A9368D18D0A91B8D11D0A9088D16D04C40AA489848AD18D0C9360E77 >4252
6753 DATAF00BA007B9F8CF9955AA8810F768A86860489848A007B955AA99F8CF99F8CB8811EC >4044
6785 DATA10F468A86860FFFFFFFFFFFFFFFFA90085FBA9D885FCA90085FDA90685FEA00014FC >3679
6817 DATAB1FD48B1FB0A0A0A0A8D3C03C8B1FB290F0D3C03B891FD68484A4A4A4A91FBC80DD0 >4658
6849 DATA6891FBE6FDD002E6FE18A5FB690285FB9002E6FCA5FCC9DCD0C4ADF80349808D147C >3883
6881 DATAF803ADC0AA48AD21D08DC0AA688D21D06000A515D071A514C9A0B06B901A206B0F9F >4171
6913 DATA99E0C8B0622CF80330E8A515F00AC902B055A514C940B04F8EFC03A5148DFA030FA7 >3835
6945 DATAA5158DFB0360202CAB18986DFA038D3C038A6DFB038D3D03204599202CAB18980BE9 >4893
6977 DATA6DFC038D3E038A6900C900D01AAE3E03AD3C038514AD3D0385154CD0AA202D990B87 >4783
7009 DATAA23C20D898AA604C28A920730020F5AA4C48AB20730020CDAA20359920AAAB2C0CDF >4351
7041 DATAF80310158A4AAAA5FECA30054A4A4C58AB2903A8A9004C53A1A5FE25FDD003A00E18 >4187
7073 DATA002CA001A9004C53A120F5AA4C81AB20CDAA20AAAB2CF803100E8A4AAAA5FE3D0D9C >4261
7105 DATA47A91D43A991FB60A5FD2CF903300505FE91FB6049FF25FE91FB602CF80310100F71 >4508
7137 DATA0EFA032EFB0320BFAB4EFB036EFA0360ACFC03984A4A4AAABD16AC85FC8A29030E59 >4641
7169 DATAAABD30AC85FB9829071865FB85FBADFA0329F885FEA9E005FC85FC18A5FB65FE12F7 >3817
7201 DATA85FBA5FC6DFB0385FCADFA0329074907AABD34AC85FD78A9348501A000B1FBB510AC >3973
7233 DATAFEA93685015860000102030506070B0A0B0C0D0F10111214151617191A1B1C1E0489 >5170
7265 DATA1F0040B0C0010204081020408 0ADFA038DFD03ADFB038DFE03ADFC038DFF03200C69 >4560
7297 DATAF5AA4C69ACADFA038DFD03ADFB038DFE03ADFC038DFF0320CDAA2081AB865E7810EA >4172
7329 DATAA9348501ADFD0338EDFA0348AEFE038AEDFB03B55AB00A6849FF690148A900E50F27 >4396
7361 DATA5A855B855C688557855BADFF0318EDFC03900449FF69FE8559665A38E557AAA90F97 >4919
7393 DATAFFE558855DB0112CF80330070A2059AD38B0050A203FAD38A55B6559B55BA55C0C47 >4804
7425 DATAE900855C08A000B1FB85FE865FA55E0AAA2084ABA65F28E8D00AE65DD006A9360F73 >4113
7457 DATA85015860A55AB0BF2007AD18A55B6557855BA55C655850CA301AA5FB2907F0050D1B >4492
7489 DATA18A9FF9004A9C7C6FC65FB85FBA5FCE90085FC60A5FB2907C907F00538A900B011F7 >3650
7521 DATA04A938E6FC65FB85FBA90065FC85FC60103EE65EA55EC9049023A900855EA5FB10D3 >4124
7553 DATA38E90885FBB016C6FC60101206FD900D26FDA5FB38E90885FBB002C6FC6046FD10D6 >4027
7585 DATA900D66FDA5FB18690B85FB9002E6FC60C65E10FBA903855EA5FB18690885FB901074 >4508
7617 DATAEEE6FC6020CDAAA00020C9AD20459920CDAAA00320C9ADA017A2058E4203BEDC0F96 >3949
7649 DATAADBD3C03AE42039DFA038BCE42031 0EE984B2069AC68A810E060ADFA03993C030DC6 >4532
7681 DATAADFB03993D03ADFC03993E036003040503040203040200010200010200010500059 4 >4821
7713 DATA01050304052 0CDAAA00020C9AD20459920CDAAA00320C9AD20459920CDAAA0060BE8 >3974
7745 DATA20C9AD18AD42036D3F038D4503AD43036D40038D460338AD4503ED3C038D450309CB >4820
7777 DATAAD4603ED3D038D460318AD44036D4103AAA9006900A8388AED3E038D470398E90B62 >4769
7809 DATA00D04DAD4703C9C8B0462CF8033012AD4603F019C902B038AD4503C940B031900D25 >4175
7841 DATA0CAD4603D02AAD4503C9A0B023A047A205BE4803BEA2AEBD3C03AE48039DFA030D31 >4194
7873 DATA88CE480310EE98482069AC68A810E0604C28A900010200010500010503040503084F >4879
7905 DATA04050304020304020001020001020607080607080 90A08090A08090A0B090A0B00BD >5283
7937 DATA06070B06070B06070806070B000105030405090A0B03040209 0A0820CDAAA00002E8 >4806
7969 DATA20C9AD20459920CDAAA00320C9ADAD3E03CD41039013A002B93C0348B93F03990C7C >4257
8001 DATA3C0368993F038810EFA005BE35AFBD3C0399FA038810F42069ACEE3E03AD41030CC3 >4461
8033 DATACD3E03B0E460000102030402 4C28A920CDAA204599205599E0A6B0F08E5203200BF7 >4354
8065 DATA4599205599E0A6B0E38E53038E4103ADFA038D3C03ADFB038D3D03ADFC038D3E0E20 >4673
8097 DATA03A9008D42038D3F038D40038563856B8562856AAD5303856185692058B18D460B9E >5369
8129 DATA038E47038C48038D49038E4A038C4B038D4F038E50038C51030E46032E47032E07DD >5602
8161 DATA4803A90085638568856285 6AAD5203856185692058B18D43038E44038C45038D0BAA >5308
8193 DATA4C038E4D038C4E030E43032E44032E450318 6D4F038D4F038A6D50038D50039807C1 >5396
8225 DATA6D51038D5103A9008563856 2AD53038561AD43038569AD4403856AAD4503856B0BAC >4899
8257 DATA2058B138ED4C038D4C038AED4D038D4D0398ED4E038D4E0320D6B02042B120D60C80 >4774
8289 DATAB0202CB120D6B02042B120D6B0202CB1AD5103304038AD4F03ED49038D4F03AD0C76 >4274
8321 DATA5003ED4A038D5003AD5103ED4B038D51031BAD49036D46038D4903AD4A036D4709D8 >4996
8353 DATA038D4A03AD4B036D48038D4B03EE3F03D003EE4003AD5103103B18AD4F036D4C09BB >4954
8385 DATA038D4F03AD50036D4D038D5003AD51036D4E038D510338AD4C03ED43038D4C030962 >5276
```

Listing 4/6.4.3.3 (Teil 4)

```
8417 DATAAD4D03ED44038D4D03AD4E03ED45038D4E03CE4103AD4103F0034C27B020D6B00BDE >4600
8449 DATA2042B120D6B06018AD3C036D3F038DFA03AD3D036D40038DFB0318AD3E036D410B32 >4668
8481 DATA03BDFC03A9006D4203D04720FEB060ADFC03C9C8B03C2CF8033013ADFB03F00B0E68 >4090
8513 DATAC902B02EADFA03C940B0274C81A8ADFB03D01FADFA03C9A0B0184C81AB18AD410F9E >3872
8545 DATA0349FF69018D4103AD420349FF69008D42036018AD3F0349FF69018D3F03AD400AD0 >4874
8577 DATA0349FF69008D400360A900856C856D856EA218066C266D266E066126622663900ABE >5212
8609 DATA1318A56C6569856CA56D656A856DA56E656B856ECAD0DCA56CA66DA46E6020CD0F9D >4799
8641 DATAAAADFA038D3C03ADFB038D3D03ADFC038D3E032045992055998E4003204599200C0D >5024
8673 DATA55998E4103204599205D99A514A6158D42038E4303204599205D99A514A6158D0B03 >5254
8705 DATA44038E4503EC4303F015B01A18AD44036968 8D4403AD45036901 8D45039007CD0A67 >4815
8737 DATA4203F0E890E62045992055998E4603A9008D3F03205B822C3F0330082081ABCE0BDB >4681
8769 DATA3F0330032069ACAD4203CD4403D008AD4303CD4503F03418AD42036D46038D420A43 >4642
8801 DATA03AD430369008D4303CD4503F01190C4AD44038D4203AD45038D43034C03B2AD0AD8 >4592
8833 DATA4203CD4403F0F590F3B0E560ADFA038DFD03ADFB038DFE03ADFC038DFF03AC42114F >4084
8865 DATA03AD43032053A1A97BA09DA23220D898A23820D898A23420D89BA900AC4003200D57 >4316
8897 DATA53A1A23620D898A23C20D89818AA986D3C038DFA038A6D3D038DFB0330732CF80DE3 >4730
8929 DATA033011C900F018C902B06FADFA03C940B068900BC900D06DADFA03C9A0B066AC0F40 >4028
8961 DATA4203AD43032053A1A97BA09DA23220D898A23A20D89BA23420D89BA900AC41030D7C >4188
8993 DATA2053A1A23620D898A23C20D89B388414B515AD3E03E5148DFC03A900E51530 0B0CA5 >4327
9025 DATAADFC03C9C8B00160A9C72CA9008DFC0360A9008DFA038DFB0360A93F8DFA03A90FB8 >3982
9057 DATA018DFB0360A99F8DFA03A9008DFB03604C28A92055998E3C03E028B0F32045990DF3 >4610
9089 DATA2055998E3D03E019B0E62045992055998E3E03EC3C0390D8E028B0D4204599200D83 >4786
9121 DATA55998E3F03EC3D0390C6E019B0C22045992055 99E010B0B886FC2045992055990E9D >4747
9153 DATAE0108AB0AB06FC06FC06FC06FC05FC48AD3D030AA818A90079E0B385FBA9C8790FFC >4020
9185 DATAE1B385FC68AC3E0391FB883005CC3C03B0F6CE3F03AE3F033016EC3D0390114 80DBF >4539
9217 DATA18A5FB692885FBA5FC690085FC6890D5600000280050007800A000C800F000180CE1 >4261
9249 DATA014001680190018801E00108023002580280 02A802D002F80220034803700398 07DD >4634
9281 DATA03C003--00C6                          >755
```

Listing 4/6.4.3.3 (Teil 5)

4/6.4.4

Grafikbefehle (Basic-Erweiterung, Version 1.2)

In der Version 1.2 unserer Basic-Erweiterung wollen wir uns einen Satz von Befehlen zur Verfügung stellen, die dem Arbeiten mit der hochauflösenden Grafik, sowie der Mehrfarbengrafik dienen. Dabei wurde darauf geachtet, das die bereitgestellten Befehle möglichst in beiden Grafikauflösungen funktionieren. Nur die Spezialbefehle zum Setzen der Farben oder Einschalten der Grafik benötigen verschiedene Aufrufe.

Mit den Grundlagen der Grafikprogrammierung werden wir uns hier nicht beschäftigen. Diese können Sie an anderer Stelle nachlesen. Für das Verständnis der folgenden Assemblerroutinen sind diese Grundlagen aber erforderlich. Um die Befehle zu verwenden, benötigen sie die Grundlagen natürlich nicht unbedingt, sie helfen einem aber doch häufig beim Lösen von eigenen Problemstellungen.

Die Version 1.2 wird folgende zusätzlichen Befehle zur Verfügung stellen:

Befehl	Bedeutung
DRAWMODE X	1=Punkt setzen; 0=Punkt löschen (in hochauflösender Grafik)
BORDER X	Rahmenfarbe 0 bis 15 festlegen
PEN X	Zeichenfarbe 0 bis 3 in Multicolorgrafik festlegen
INK X	Textfarbe 0 . .15 festlegen
PAPER X	Hintergrundfarbe 0 . .15 Text festlegen
GRAFON ZF,HF	hochauflösende Grafik ein; Zeichenfarbe ZF; Hintergrund HF
MULTION HF,F1,F2,F3	Multicolorgrafik an; Hintergrund HF, Farbe 1 bis 3 = F1 bis F3
GRAFCLR	Grafik löschen
GRAFOFF	Textmodus einschalten

Befehl	Bedeutung
MOVE X,Y	Grafikcursor setzen
MOVER X,Y	Grafikcursor setzen (relativ)
PLOT X,Y	Punkt setzen
PLOTER X,Y	Punkt setzen (relativ)
LINE X,Y	Linie ziehen
LINER X,Y	Linie ziehen (relativ)
FRAME X1,Y1,X2,Y2	Rahmen zeichnen zwischen (X1,Y1) und (X2,Y2)
BLOCK X1,Y1,X2,Y2	ausgefüllter Rahmen (Rechteck)
CIRCLE X,Y,RX,RY	Ellipse mit Mittelpunkt (X,Y) sowie Radien RX und RY
ANGLE X,Y,RX,RY,AW,EW,SW	Kreissegment zeichnen
A=TEST(X,Y)	Punktfarbe des Punktes (X,Y) liefern
A=TESTR(X,Y)	Punktfarbe liefern (relativ)
BOX XO,YO,XU,YU,XH,YH	Quader zeichnen
COLBLOCK X1,Y1,X2,Y2	Farbbereich festlegen

An dieser Stelle wollen wir auch gleich ein kleines Demonstrationsprogramm vorstellen, welches die wesentlichen Befehle der Grafikerweiterung zeigt. Es besteht aus einer Sammlung kleiner Unterprogramme, welche verschiedene Grafiken auf den Bildschirm zaubern. Diese werden vom Hauptprogramm nacheinander aufgerufen. Sie können die Unterprogramme natürlich auch einzeln eingeben und ablaufen lassen.

```
10 print"XBASIC GRAPHIK DEMO":print:print
20 forqq=1to8
30 reada$:printa$:pause5:grafclr
40 onqqgosub4000,1000,4500,5000,3000,2000,5500,6000
60 pause5:grafoff:grafclr
70 next
80 end
999 rem **** demo: line
1000 dimx%(24),y%(24)
1010 grafclr:grafon7,2
1020 fort=3to25:pause2
1030 wi=2*3.14159/t
1040 fors=0tot-1
1050 x%(s)=sin(wi*s)*159+160
1060 y%(s)=cos(wi*s)*99+100
1070 next:grafclr
1080 fors=0tot-2
1090 forss=s+1tot-1
1100 movex%(s),y%(s):linex%(ss),y%(ss)
1110 next:next
1120 next:return
1999 rem **** demo: angle
2000 grafon1,0:grafclr
2010 forx=5to100step2
2020 angle40+x*.8,100,x*1.2,x,315,90,45
2030 angle40+x*.8,100,x*1.2,x,135,270,45
2040 next
2050 angle120,100,120,100,0,360,45
2060 return
2999 rem **** demo: circle
3000 multion2,7,1,14:grafclr:pe=1
3010 fora=5to165step10:forb=5to165step10
3020 circle80,100,a,b
3030 penpe:pe=pe+1:ifpe=4thenpe=1
3040 next:next
3050 return
3999 rem **** demo: plot
4000 multion0,1,7,14
4010 fort=0to400:plot160*rnd(8),200*rnd(8):pen4*rnd(8):next
4020 return
4499 rem **** demo: frame
4500 grafon7,0:fort=0to160step5
4510 framet,t,319-t,199-t:next:return
4999 rem **** demo: block
5000 multion2,7,14,1
5010 pen1:block10,10,40,180
5020 pen2:block50,10,80,180
5030 pen3:block90,10,120,180
5040 return
5499 rem **** demo: box
5500 grafon1,0
5510 box10,120,200,190,50,10
5520 return
6999 rem **** demo: colblock
6000 grafon7,2:drawmode0
6010 fort=0to319step5
6040 movet,0:line319-t,199
6050 next
6060 fort=0to199step5
6090 move0,t:line319,199-t
6100 next
6110 pause10
6120 fort=0to300
```

Listing 4/6.4.4-1 (Teil 1)

```
6130 x=rnd(8)*39
6140 y=rnd(8)*24
6150 x2=x+rnd(8)*(39-x)+1
6160 y2=y+rnd(8)*(24-y)+1
6170 colblockx,y,x2,y2,15*rnd(8),15*rnd(8)
6180 next
6190 block0,0,319,199
6200 fort=0to300
6210 x=rnd(8)*39
6220 y=rnd(8)*24
6230 x2=x+rnd(8)*(39-x)+1
6240 y2=y+rnd(8)*(24-y)+1
6250 colblockx,y,x2,y2,15*rnd(8),15*rnd(8)
6260 next
6270 pause7:grafclr:return
6999 rem **** liste der befehle
7000 data"PLOT","LINE","FRAME"
7001 data"BLOCK","CIRCLE","ANGLE","BOX"
7002 data"COLBLOCK"
```

Listing 4/6.4.4-1 (Teil 2)

In der Version 1.2 werden wir wieder einige Programmteile verändern oder ergänzen müssen. Das Hauptproblem ist, das die Speicherplatzreserven, die in der Version 1.0 für kommende Erweiterungen definiert wurden, schneller als erwartet erschöpft sind. Sie wurden bereits von der Version 1.1 weitgehend verbraucht. Mit der Version 1.2 werden wir uns deshalb Platz verschaffen müssen, und uns bei dieser Gelegenheit neue Reserven anlegen.

Dazu müssen wir leider wieder etwas von dem Basic-Speicherplatz für unsere Erweiterung reservieren. Aber wir begnügen uns noch mit weiteren 0.5 KByte, so daß insgesamt nur 2 KByte des Basic-Speicherplatzes verlorengehen. Dieser Speicherplatzverlust wird durch die zusätzlichen Befehle aber leicht wieder ausgeglichen, da diese nur einen Bruchteil vergleichbarer Basic-Unterprogramme benötigen.

Weil der Quelltext ebenfalls sehr lang geworden ist, sollten Sie ihn auf verschiedene Dateien verteilen, welche dann mit dem Pseudobefehl „.FI" nacheinander assembliert werden.

Die Version 1.2 benötigt folgende zusätzliche bzw. veränderte Symbole:

Für eine 24-Bit Multiplikation benötigen wir:

```
reg1:     .eq 97          ;1. Operand fuer Multiplikation
reg2:     .eq 105         ;2. Operand
merg:     .eq 108         ;Ergebnis
```

Der Block für den Line-Input Befehl wird vom Programmende in die ungenutzte alte Bildschirmseite verlegt. Dazu wird die Labeldefinition am Ende des Line-Input# -Befehls entfernt, und das folgende Symbol eingeführt:

```
inpuffer:      .eq  $500           ;Block fuer Line-Input
```

Für die Grafikbefehle benötigen wir einige Speicherstellen, welche die aktuellen Bildschirmkoordinaten, die eingestellte Auflösung sowie den Zeichenmodus bestimmen:

```
ycurs2:        .eq  1023           ;Grafik Endpunkt Y
xcurs2:        .eq  1021           ;Grafik Endpunkt X
drawmode:      .eq  1017           ;Zeichenmodus
xcurs:         .eq  1018           ;Grafik X
ycurs:         .eq  1020           ;Grafik Y
colmode:       .eq  1016           ;Grafik Farbmodus
```

Da bei der Farbgrafik der Farbspeicher benötigt wird, müssen wir diesen für die Textseite retten. Dazu wird ein Block von 512 Byte definiert:

```
colmem:        .eq  $600           ;Farbenspeicher
```

Um bei einer Fehlermeldung, oder am Programmende automatisch auf die Textseite umschalten zu können, müssen wir uns außerdem in die Eingabe-Warteschleife einbauen:

```
eingabevec:    .eq  $0302          ;Vektor fuer Eingabe-Warteschleife
oldeingabe:    .eq  $A483          ;alte Eingabe-Warteschleife
```

Schließlich wird noch der Start der Basicerweiterung um 0.5 KByte vorverlegt:

```
xbasicstart:   .eq  $A000-2048     ;Startadresse von XBASIC
```

Wie bereits erwähnt, wollen wir bei Fehlermeldungen oder am Programmende auf die normale Textseite umschalten. Dazu bietet es sich an, eine eigene Routine in die Eingabe-Warteschleife einzubauen, denn diese wird in beiden Fällen vom normalen C64-Basic aufgerufen. Außerdem wird sie über einen Vektor in der Zero-Page aufgerufen, so daß wir hier leicht eine eigene Routine einbauen können.

Wir müssen also in die Initialisierungsroutine der Basic-Erweiterung eine Umleitung des Vektors einbauen. Zusätzlich müssen die neuen, von der Grafik benötigten Symbole auf Anfangswerte gesetzt werden. Das folgende Listing zeigt die veränderte Initialisierungsroutine:

```
9817 a9 36      initxbasic:lda #%00110110    ;Adresse des Bildschirms
9819 8d 18 d0              sta vic+24        ;und des Zeichensatzes festlegen
981c ad 00 dd              lda cia2          ;VIC Bereich von $C000-$FFFF
981f 29 fc                 and #%11111100
9821 8d 00 dd              sta cia2
9824 a9 cc                 lda #textseite/256 ;Textseite fuer Basic festlegen
9826 8d 88 02              sta charpage

9829 a9 00                 lda #<xbasicstart  ;Neues Basic-Ende setzen
982b a2 98                 ldx #>xbasicstart
982d 85 37                 sta basend
982f 86 38                 stx basend+1
9831 a9 a9                 lda #<newnmi      ;neue NMI-Routine
9833 a2 98                 ldx #>newnmi
9835 8d 18 03              sta nmivec
9838 8e 19 03              stx nmivec+1
983b a9 29                 lda #<token       ;neue Token-Wandelroutine
983d a2 9a                 ldx #>token
983f 8d 04 03              sta tokenvec
9842 8e 05 03              stx tokenvec+1
9845 a9 ac                 lda #<list        ;neue List-Routine
9847 a2 9e                 ldx #>list
9849 8d 06 03              sta listvec
984c 8e 07 03              stx listvec+1
984f a9 71                 lda #<execute     ;neue Ausfuehrungsroutine
9851 a2 9f                 ldx #>execute
9853 8d 08 03              sta executevec
9856 8e 09 03              stx executevec+1
9859 a9 45                 lda #<exefkt      ;Neues Basic-Ende setzen
985b a2 9f                 ldx #>exefkt
985d 8d 0a 03              sta fktvec
9860 8e 0b 03              stx fktvec+1
9863 a9 99                 lda #<neweingabe  ;Neue Eingabewarteschleife
9865 a2 98                 ldx #>neweingabe
9867 8d 02 03              sta eingabevec
986a 8e 03 03              stx eingabevec+1
986d a9 00                 lda #0            ;Grafik auf Standard
986f 8d f9 03              sta drawmode
9872 8d f8 03              sta colmode
9875 60                    rts
```

Listing 4/6.4.4-2

In die neue Eingabe-Warteschleife wird dann nur noch der Aufruf der Routine <dografoff>eingebaut. Diese Routine schaltet auf den Textbildschirm um.

```
                     ;-----------------------------------
                     ;---   XBASIC Eingabe-Warteschleife   ---
                     ;-----------------------------------
9899 a5 01           neweingabe:lda pp
989b 48                         pha
989c a9 36                      lda #$36
989e 85 01                      sta pp
98a0 20 08 aa                   jsr dografoff
98a3 68                         pla
98a4 85 01                      sta pp
98a6 4c 83 a4                   jmp oldeingabe
```

Listing 4/6.4.4-2 (Teil 2)

Für die Grafikroutinen benötigen wir außerdem einige zusätzliche ROM-Befehle, da wir für den ANGLE-Befehl die Fließkommaroutinen des Sinus und des Cosinus, sowie die Routine für Multiplikationen verwenden werden. Die Eingabe von relativen Koordinaten erfordert ebenfalls einen Eintrag in die Tabelle, da unsere bisherigen Routinen nur das Einlesen von positiven Zahlen vorsehen. Die vollständige Tabelle ist hier nochmals abgebildet:

```
                     rombefehle:
9989 fe ae                      .wo testcode-1      ;0
998b 25 b5                      .wo garbage-1       ;2
998d 81 b7                      .wo getstring-1     ;4
998f f6 b8                      .wo $b8f7-1         ;6
9991 48 bc                      .wo $bc49-1         ;8
9993 3a ab                      .wo CURrechts-1     ;10
9995 e3 ff                      .wo get-1           ;12
9997 6b e5                      .wo $e56c-1         ;14
9999 a5 b6                      .wo frestr-1        ;16
999b 32 a5                      .wo 42291-1         ;18
999d 8a b0                      .wo varsuch-1       ;20
999f dc bd                      .wo $bddd-1         ;22
99a1 7b a5                      .wo $a57c-1         ;24
99a3 b9 ff                      .wo $ffba-1         ;26 Fileparameter
99a5 bc ff                      .wo $ffbd-1         ;28 Filename
99a7 bf ff                      .wo $ffc0-1         ;30 Open
99a9 c2 ff                      .wo $ffc3-1         ;32 Close
99ab c5 ff                      .wo $ffc6-1         ;34 Chkin
99ad c8 ff                      .wo $ffc9-1         ;36 Chkout
99af cb ff                      .wo $ffcc-1         ;38 Clrch
99b1 ce ff                      .wo $ffcf-1         ;40 Basin
99b3 d1 ff                      .wo bsout-1         ;42
99b5 28 9a                      .wo token-1         ;44
99b7 6a a9                      .wo $a96b-1         ;46
99b9 e0 ff                      .wo $ffe1-1         ;48 STOP-Taste
99bb 27 ba                      .wo $ba28-1         ;50 FAC*(A/Y)
```

Listing 4/6.4.4-2 (Teil 3-1)

```
99bd 0b bc                .wo $bc0c-1        ;52 FAC->ARG
99bf 2a ba                .wo $ba2b-1        ;54 FAC:=FAC*ARG
99c1 6a e2                .wo $e26b-1        ;56 sin
99c3 63 e2                .wo $e264-1        ;58 cos
99c5 a9 b1                .wo $b1aa-1        ;60 FAC nach Integer

              RESERVE1:   .eq 80*2+rombefehle-RESERVE1
                          .db RESERVE1
Reserve Rombefehle:
                          .tx "Reserve Rombefehle:"
0062
                          .oo RESERVE1
```

Listing 4/6.4.4-2 (Teil 3-2)

Schließlich müssen wir die neuen Befehle in die Befehlsliste eintragen. Beachten Sie, daß TEST und TESTR Funktionen sind. Daher verschiebt sich auch der Dummy-Eintrag für den Befehl mit der Nummer 32. Bei der Gelegenheit werden wir uns auch gleich eine zusätzliche Fließkommakonstante definieren, die den Namen „pifac" tragen wird. Diese Konstante (pifac = $2* \pi / 360 \approx 0.01745$) wird von dem Befehl ANGLE zur Umrechnung von Winkeln in Gradangabe, in eine Darstellung in Radiant benötigt, da die Funktionen Sinus und Cosinus die Winkelangabe in Radiant erwarten.

```
              ;-------------------------------------
              ;---  Liste der XBASIC Befehle     ---
              ;-------------------------------------
              ;  1. Liste der Funktionen
9ab9 10       funktionen:.by 15+1        ;Anzahl der Funktionen +1

9aba b8 28 00 beflist:   .by 184,'(,0   ;FRE(  Liste der Funktionsworte
9abd 41 54 28            .by "at(",0    ;AT(
00
9ac1 48 45 58            .by "hex$",0   ;HEX$
24 00
9ac6 44 45 43            .by "dec",0    ;DEC
00
9aca 49 4e 4b            .by "inkey$",0 ;INKEY$
45 59 24 00
9ad1 49 4e 53            .by "instr(",0 ;INSTR(
54 52 28 00
9ad8 53 54 52            .by "string$(",0 ;STRING$
49 4e 47 24 28 00
9ae1 53 50 41            .by "space$(",0 ;SPACE$
43 45 24 28 00
9ae9 44 52 45            .by "dreek",0  ;DREEK
45 4b 00
9aef 52 45 45            .by "reek",0   ;REEK
4b 00
9af4 44 45 45            .by "deek",0   ;DEEK
4b 00
```

Listing 4/6.4.4-3 (Teil 1)

```
9af9 53 54 24          .by "st$",0          ;ST$
00
9afd 54 49 4d          .by "time",0         ;TIME
45 00
9b02 54 45 53          .by "testr(",0       ;TESTR
54 52 28 00
9b09 54 45 53          .by "test(",0        ;TEST
54 28 00

                ;  2. Liste der Befehle
9b0f 43 4c 53          .by "cls",0          ;CLS  Liste der Befehlsworte
00
9b13 4f 4c 44          .by "oldchar",0      ;OLDCHAR
43 48 41 52 00
9b1b 43 36 34          .by "c64",0          ;C64
00
9b1f 58 42 41          .by "xbasic",0       ;XBASIC
53 49 43 00
9b26 49 4e 53          .by "inst",0         ;INST
54 00
9b2b 96 43 48          .by 150,"char",0     ;DEFCHAR
41 52 00
9b31 4f 4c 44          .by "old",0          ;OLD
00
9b35 44 52 4f          .by "droke",0        ;DROKE
4b 45 00
9b3b 52 4f 4b          .by "roke",0         ;ROKE
45 00
9b40 44 4f 4b          .by "doke",0         ;DOKE
45 00
9b45 41 55 a4          .by "au",164,0       ;AUTO
00
9b49 53 57 41          .by "swap",0         ;SWAP
50 00
9b4e 44 49 52          .by "dir",0          ;DIR
00
9b52 51 93 00          .by 'q,147,0         ;QLOAD
9b55 43 4f 4c          .by "collect",0      ;COLLECT
4c 45 43 54 00
9b5d 48 45 41          .by "header",0       ;HEADER
44 45 52 00
9b64 44 49 52          .by "dir",0          ;Dummy fuer Befehl 32
00
9b68 52 45 4e          .by "rename",0       ;RENAME
41 4d 45 00
9b6f 53 43 52          .by "scratch",0      ;SCRATCH
41 54 43 48 00
9b77 49 4e 49          .by "initialise",0 ;INITIALISE
54 49 41 4c 49 53 45 00
9b82 53 45 54          .by "setd",193,'r,0;SETDATNR
44 c1 52 00
9b89 50 41 55          .by "pause",0        ;PAUSE
53 45 00
9b8f 53 45 54          .by "settime",0      ;SETTIME
54 49 4d 45 00
9b97 53 45 54          .by "seteol",0       ;SETEOL
45 4f 4c 00
9b9e 4c 49 4e          .by "line",133,0     ;LINEINPUT
45 85 00
9ba4 4c 49 4e          .by "line",132,0     ;LINEINPUT#
45 84 00
9baa 44 52 41          .by "drawmode",0     ;DRAWMODE
57 4d 4f 44 45 00
9bb3 42 b0 44          .by 'b,176,"der",0 ;BORDER
```

Listing 4/6.4.4-3 (Teil 2)

```
45 52 00
9bb9 49 4e 4b                  .by "ink",0         ;INK
00
9bbd 50 41 50                  .by "paper",0       ;PAPER
45 52 00
9bc3 50 45 4e                  .by "pen",0         ;PENCIL
00
9bc7 47 52 41                  .by "graf",145,0    ;GRAFON
46 91 00
9bcd 47 52 41                  .by "grafoff",0     ;GRAFOFF
46 4f 46 46 00
9bd5 47 52 41                  .by "graf",156,0    ;GRAFCLR
46 9c 00
9bdb 4d 55 4c                  .by "multi",145,0   ;MULTION
54 49 91 00
9be2 4d 4f 56                  .by "mover",0       ;MOVER
45 52 00
9be8 4d 4f 56                  .by "move",0        ;MOVE
45 00
9bed 50 4c 4f                  .by "plotr",0       ;PLOTR
54 52 00
9bf3 50 4c 4f                  .by "plot",0        ;PLOT
54 00
9bf8 4c 49 4e                  .by "liner",0       ;LINER
45 52 00
9bfe 4c 49 4e                  .by "line",0        ;LINE
45 00
9c03 46 52 41                  .by "frame",0       ;FRAME
4d 45 00
9c09 42 4f 58                  .by "box",0         ;BOX
00
9c0d 43 49 52                  .by "circle",0      ;CIRCLE
43 4c 45 00
9c14 41 4e 47                  .by "angle",0       ;ANGLE
4c 45 00
9c1a 43 4f 4c                  .by "colblock",0    ;COLBLOCK
42 4c 4f 43 4b 00
9c23 42 4c 4f                  .by "block",0       ;BLOCK
43 4b 00
9c29 00                        .by 0               ;Tabellenabschluss
          RESERVE2:  .eq 700+beflist-RESERVE2
                     .db RESERVE2
Reserve fuer Befehle (Text):
                     .tx "Reserve fuer Befehle (Text):"
014c
                     .oo RESERVE2
9d76 87 48 00   fk100:     .by $87,$48,0,0,0
00 00
9d7b 7b 0e fa   pifac:     .by $7b,$0e,$fa,$35,$12
35 12
          ;Tabelle der Adressen der Befehle & Funktionen
9d80 31 a1      befehle:   .wo BBfre-1         ;FRE(  * Funktionen
9d82 1f a2                 .wo BBat-1          ;AT(
9d84 bc a1                 .wo BBhex-1         ;HEX$
9d86 64 a1                 .wo BBdez-1         ;DEC
9d88 f5 a1                 .wo BBinkey-1       ;INKEY$
9d8a d3 a2                 .wo BBinstr-1       ;INSTR
9d8c 6c a2                 .wo BBstring-1      ;STRING$
9d8e 50 a2                 .wo BBspaces-1      ;SPACES
9d90 37 a5                 .wo BBdreek-1       ;DREEK
```

Listing 4/6.4.4-3 (Teil 3)

```
9d92 57 a5          .wo BBreek-1         ;REEK
9d94 74 a5          .wo BBdeek-1         ;DEEK
9d96 4e a6          .wo BBst-1           ;ST$
9d98 81 a7          .wo BBtime-1         ;TIME
9d9a 38 ab          .wo BBtestr-1        ;TESTR
9d9c 41 ab          .wo BBtest-1         ;TEST

9d9e 43 e5          .wo clrscr-1         ;CLS   * Befehle
9da0 75 98          .wo setzeichen-1     ;OLDCHAR
9da2 e1 fc          .wo reset-1          ;C64
9da4 ff 97          .wo BBxbasic-1       ;XBASIC
9da6 e5 9f          .wo BBinst-1         ;INST
9da8 53 a0          .wo BBdefchar-1      ;DEFCHAR
9daa cc 9f          .wo BBold-1          ;OLDCHAR
9dac 10 a5          .wo BBdroke-1        ;DROKE
9dae d3 a4          .wo BBroke-1         ;ROKE
9db0 e7 a4          .wo BBdoke-1         ;DOKE
9db2 78 a4          .wo BBauto-1         ;AUTO
9db4 94 a5          .wo BBswap-1         ;SWAP
9db6 e6 a5          .wo BBdir-1          ;DIR
9db8 5e a3          .wo BBqload-1        ;QLOAD
9dba 83 a6          .wo BBcollect-1      ;COLLECT
9dbc b5 a6          .wo BBheader-1       ;HEADER
9dbe 00 00          .wo 0                ;Dummy fuer Befehl 32
9dc0 db a6          .wo BBrename-1       ;RENAME
9dc2 b8 a6          .wo BBscratch-1      ;SCRATCH
9dc4 ac a6          .wo BBinitial-1      ;INITIALISE
9dc6 de a5          .wo BBsetdatnr-1     ;SETDATNR
9dc8 34 a7          .wo BBpause-1        ;PAUSE
9dca ea a7          .wo BBsettime-1      ;SETCLOCK
9dcc 7a a8          .wo BBseteol-1       ;SETEOL
9dce 82 a8          .wo BBlineinput-1    ;LINEINPUT
9dd0 d1 a8          .wo BBlineindev-1    ;LINEINPUT#
9dd2 18 a9          .wo BBdrawmode-1     ;DRAWMODE
9dd4 f7 a8          .wo BBborder-1       ;BORDER
9dd6 0d a9          .wo BBink-1          ;INK
9dd8 02 a9          .wo BBpaper-1        ;PAPER
9dda 2c a9          .wo BBpencil-1       ;PENCIL
9ddc 73 a9          .wo BBgrafon-1       ;GRAFON
9dde 07 aa          .wo BBgrafoff-1      ;GRAFOFF
9de0 4a a9          .wo BBgrafclr-1      ;GRAFCLR
9de2 c0 a9          .wo BBmultion-1      ;MULTION
9de4 f4 aa          .wo BBmover-1        ;MOVER
9de6 cc aa          .wo BBmove-1         ;MOVE
9de8 77 ab          .wo BBplotr-1        ;PLOTR
9dea 7d ab          .wo BBplot-1         ;PLOT
9dec 3b ac          .wo BBliner-1        ;LINER
9dee 53 ac          .wo BBline-1         ;LINE
9df0 92 ad          .wo BBframe-1        ;FRAME
9df2 f3 ad          .wo BBbox-1          ;BOX
9df4 3d af          .wo BBcircle-1       ;CIRCLE
9df6 8c b1          .wo BBangle-1        ;ANGLE
9df8 41 b3          .wo BBcolblock-1     ;COLBLOCK
9dfa e9 ae          .wo BBblock-1        ;BLOCK

          RESERVE3: .eq 300+befehle-RESERVE3
                    .db RESERVE3
Reserve fuer Befehle (Tab ):
                    .tx "Reserve fuer Befehle (Tab ):"
00b0
                    .oo RESERVE3
```

Listing 4/6.4.4-3 (Teil 4)

In der Version 1.1 hatten wir die Token-Wandelroutine bereits auf Tabellen über 256 Byte Länge eingestellt, da sich die Tabelle dieser Grenze rasch näherte. In dieser Version wird diese Länge nun endgültig überschritten. Daher müssen wir auch den LIST-Befehl an die veränderten Bedingungen anpassen. Dies ist aber keinesfalls schwer:

```
                ;-------------------------------------------------
                ;---  Neue LIST Routine                        ---
                ;-------------------------------------------------
9eac 10 64      list:       bpl normlist1       ;kein Token
9eae 48                     pha                 ;azg auf befehlsliste setzen
9eaf a9 ba                  lda #<beflist
9eb1 85 fb                  sta azg
9eb3 a9 9a                  lda #>beflist
9eb5 85 fc                  sta azg+1
9eb7 68                     pla
9eb8 c9 d2                  cmp #xtoken
9eba d0 59                  bne normlist2       ;kein XBASIC Befehl
9ebc 24 0f                  bit $f              ;Hochkomma ?
9ebe 30 52                  bmi normlist1       ;ja
9ec0 c8                     iny                 ;XBASIC Nummer holen
9ec1 b1 5f                  lda ($5f),y
9ec3 aa                     tax                 ;Nummer in X bringen
9ec4 84 49                  sty $49             ;Y merken
9ec6 a0 00                  ldy #0              ;Befehlswort suchen
9ec8 ca         suchbef:    dex                 ;Nummer-1
9ec9 f0 13                  beq foundbef        ;Befehlswort gefunden
9ecb c8         nextbef:    iny                 ;naechstes Befehlswort suchen
9ecc b1 fb                  lda (azg),y
9ece d0 fb                  bne nextbef
9ed0 38                     sec
9ed1 98                     tya
9ed2 a0 00                  ldy #0
9ed4 65 fb                  adc azg
9ed6 85 fb                  sta azg
9ed8 90 ee                  bcc suchbef
9eda e6 fc                  inc azg+1
9edc d0 ea                  bne suchbef         ;weitersuchen
9ede b1 fb      foundbef:   lda (azg),y         ;Ausgabe des Befehlswortes
9ee0 30 08                  bmi oldnew
9ee2 f0 34                  beq normlist3
9ee4 20 47 ab               jsr $ab47           ;Ausgabe des Zeichens
9ee7 c8                     iny
9ee8 d0 f4                  bne foundbef

9eea 84 02      oldnew:     sty xmem            ;Ausgabe alter Basic-Befehle
9eec 38                     sec                 ;Tokennummer ermitteln
9eed e9 7f                  sbc #$7f
9eef aa                     tax
9ef0 a0 ff                  ldy #$ff
9ef2 ca         wsoldbef:   dex                 ;Text suchen
9ef3 f0 08                  beq suoldbef
9ef5 c8         soldbef:    iny
9ef6 b9 9e a0               lda $a09e,y
9ef9 10 fa                  bpl soldbef
9efb 30 f5                  bmi wsoldbef
9efd c8         suoldbef:   iny                 ;Befehlswort ausgeben
9efe b9 9e a0               lda $a09e,y
9f01 30 05                  bmi weiter
9f03 20 47 ab               jsr $ab47           ;ein Zeichen ausgeben
```

Listing 4/6.4.4-4 (Teil 1)

```
9f06 d0 f5               bne suoldbef
9f08 29 7f     weiter:   and #$7f         ;letzten Buchstaben
9f0a 20 47 ab            jsr $ab47        ;ausgeben
9f0d a4 02               ldy xmem         ;und weiter bei der neuen ausgabe
9f0f c8                  iny
9f10 d0 cc               bne foundbef

9f12 4c f3 a6  normlist1: jmp $a6f3       ;Ruecksprung in die normale List-Routine
9f15 4c 1c a7  normlist2: jmp $a71c
9f18 a4 49     normlist3: ldy $49
9f1a 4c f6 a6            jmp $a6f6

```

Listing 4/6.4.4-4 (Teil 2)

Wenn wir nun noch in der Basic-Einschaltmeldung die Versionsnummer von 1.1 in 1.2 ändern, können wir uns endlich den Grafikbefehlen zuwenden.

4/6.4.4.1

BORDER X

Der BORDER-Befehl ist sehr einfach. Mit der Routine <getbyte> holen wir uns einen Wert zwischen Null und 255. Dieser Wert wird nun getestet, ob er die zulässige Höchstgrenze von 15 überschreitet. In diesem Fall wird der Fehler „Illegal Quantity" geliefert. Andernfalls kann der Wert als Rahmenfarbe in das zuständige Register des VIC geschrieben werden.

```
               ;-------------------------------------
               ;--- XBASIC-Befehl: BORDER Col     ---
               ;-------------------------------------
a8f8 20 55 99  BBborder: jsr getbyte     ;Rahmenfarbe holen
a8fb e0 10               cpx #16
a8fd b0 29               bcs nocol       ;>=15 -> ERROR
a8ff 8e 20 d0            stx vic+32      ;und setzen
a902 60                  rts

```

Listing 4/6.4.4.1

4/6.4.4.2

PAPER X

Der PAPER-Befehl macht genau das Gleiche für die Hintergrundfarbe. Auch für diese Farbe ist ein Register im VIC zuständig, dem nur der entsprechende Farbwert mitgeteilt werden muß.

```
                ;-----------------------------------------
                ;--- XBASIC-Befehl:  PAPER Col         ---
                ;-----------------------------------------
a903 20 55 99   BBpaper:   jsr getbyte      ;Hintergrundfarbe holen
a906 e0 10                 cpx #16
a908 b0 1e                 bcs nocol        ;>=15 -> ERROR
a90a 8e 21 d0              stx vic+33       ;und setzen
a90d 60                    rts
```

Listing 4/6.4.4.2

4/6.4.4.3

INK X

Für die Textfarbe existiert kein Register im VIC, da die Textfarbe für jeden Buchstaben einzeln festgelegt werden kann. Die Farbinformation steht im Farbspeicher, und wird bei einer Ausgabe vom Betriebssystem des C64 in den Farbspeicher geschrieben. Die aktuelle Textfarbe merkt sich dabei das Betriebssystem in der Speicherstelle mit der Adresse 646. Diese Speicherstelle wird von INK neu belegt, so wie es BORDER und PAPER mit den Registern des VIC gemacht haben.

```
                  ;-------------------------------------------
                  ;--- XBASIC-Befehl:    INK Col           ---
                  ;-------------------------------------------
a90e 20 55 99     BBink:     jsr getbyte        ;Zeichenfarbe holen
a911 e0 10                   cpx #16
a913 b0 13                   bcs nocol          ;>=15 -> ERROR
a915 8e 86 02                stx 646            ;und setzen
a918 60                      rts
```

Listing 4/6.4.4.3

4/6.4.4.4

DRAWMODE X

Um mit den Befehlen in der hochauflösenden Grafik sowohl Punkte setzen, als auch löschen zu können, wird mit dem Befehl DRAWMODE die Arbeitsweise festgelegt. Der Befehl selbst nimmt dabei nur ein Argument von 0 oder 1 entgegen, und wandelt es in den Wert 0 bzw. 128, indem das Bit 0 in das Bit 7 rotiert wird. Dieser Wert wird als Flagge gespeichert.

```
                  ;-------------------------------------------
                  ;--- XBASIC-Befehl: DRAWMODE Mode       ---
                  ;-------------------------------------------
a919 20 55 99     BBdrawmode:jsr getbyte        ;Zeichenmodus festlegen
a91c e0 02                   cpx #2
a91e b0 08                   bcs nocol          ;DRAWMODE>1
a920 8a                      txa                ;Bit0 -> Bit 7
a921 18                      clc
a922 6a                      ror
a923 6a                      ror
a924 8d f9 03                sta drawmode       ;speichern
a927 60                      rts

a928 a2 0e        nocol:     ldx #14            ;Illegal Quantity melden
a92a 4c 28 99                jmp fehler
```

Listing 4/6.4.4.4

4/6.4.4.5

PEN X

Der PEN-Befehl legt die verwendete Farbe im Multicolormodus fest. Beim Einschalten der Multicolorgrafik werden vier Farben angegeben, eine Hintergrundfarbe und drei Zeichenfarben. Mit dem Befehl „PEN" kann nun jede der vier Farben als Zeichenfarbe ausgewählt werden.

Die Routine holt dazu zunächst einen Wert von 0 bis 3, der der gewünschten Farbe entspricht. Dieser Wert wird dann in einer Tabelle abgelegt, wobei die Farbinformation nacheinander in den Bits 0..1, 2..3, 4..5 und 6..7 steht. Diese Tabelle wird benötigt, da in einem Byte max. 4 Punkte auftreten können. Für jeden dieser Punkte steht der Farbwert in der Tabelle „penbits". Die Tabelle „delbits" enthält die Werte, mit welchen genau die zum Punkt gehörenden zwei Bits aus einem Byte per AND-Befehl ausgeblendet werden können.

```
                ;---------------------------------------
                ;--- XBASIC-Befehl: PEN Nr            ---
                ;---------------------------------------
a92d 20 55 99   BBpencil:  jsr getbyte          ;Farbnummer holen
a930 e0 04                 cpx #4               ;>3 ?
a932 b0 f4                 bcs nocol            ;Fehler melden
a934 8a                    txa                  ;In Accu
a935 a2 00                 ldx #0               ;4 mal 2*Linksschieben
a937 9d 43 a9   setpenbits:sta penbits,x        ;ergibt Masken fuer die
a93a 0a                    asl                  ;Farbe
a93b 0a                    asl
a93c e8                    inx
a93d e0 04                 cpx #4
a93f d0 f6                 bne setpenbits
a941 60                    rts

a942 01         pen:       .by 1                ;1 ist Einschaltfarbe
a943 01 04 10   penbits:   .by %01,%0100,%010000,%01000000;Masken fuer Farbe 1
40
                ;Masken fuer Ausblenden der Bits
a947 fc f3 cf   delbits:   .by 255-%11,255-%1100,255-%110000,255-%11000000
3f
```

Listing 4/6.4.4.5

4/6.4.4.6

GRAFCLR

Der Befehl GRAFCLR benötigt keine Parameter. Er löscht lediglich den 8 KByte großen Grafikbereich, der von $E000 bis $FFFF, also unter dem Kernal-ROM, liegt. Zum Löschen des Speichers werden zwei Zeiger verwendet, wobei mit dem ersten Zeiger die unteren 4 KByte, und mit dem zweiten Zeiger die oberen 4 KByte gelöscht werden. Dies beschleunigt die Arbeit etwas.

Noch schneller ginge es, wenn man den Bereich seitenweise (also in 256-Byte Schritten) löschen würde. Da dies jedoch 32 Befehle der Form „STA HIRESSEITE+xxx,Y" erfordern würde, wurde hier darauf verzichtet.

GRAFCLR initialisiert außerdem noch den Grafikcursor mit (0,0), was der linken oberen Ecke entspricht.

```
                ;-------------------------------------
                ;--- XBASIC-Befehl: GRAFCLR        ---
                ;-------------------------------------
a94b a0 00      BBgrafclr: ldy #0             ;Grafik Cursor auf Anfangspos.
a94d 8c fa 03              sty xcurs
a950 8c fb 03              sty xcurs+1
a953 8c fc 03              sty ycurs
a956 a9 e0                 lda #>hiresseite   ;azg auf Hiresseite
a958 85 fc                 sta azg+1
a95a 84 fb                 sty azg
a95c a9 f0                 lda #>hiresseite+4096;azg2 auf 2.Haelfte
a95e 85 fe                 sta azg2+1
a960 84 fd                 sty azg2
a962 a2 10                 ldx #16            ;2*16 Seiten loeschen
a964 98                    tya
a965 91 fb      grafclr:   sta (azg),y        ;zwei Seiten loeschen
a967 91 fd                 sta (azg2),y
a969 c8                    iny
a96a d0 f9                 bne grafclr
a96c e6 fc                 inc azg+1          ;bis alle Seiten geloescht
a96e e6 fe                 inc azg2+1
a970 ca                    dex
a971 d0 f2                 bne grafclr
a973 60                    rts
```

Listing 4/6.4.4.6

4/6.4.4.7

GRAFON ZF,HF

Mit dem Befehl GRAFON wird die hochauflösende Grafik eingeschaltet, und die Farben für Punkte sowie Hintergrund festgelegt.

Der Befehl programmiert dazu den VIC auf die neue Betriebsart um. Dann sind eventuell vorhandene Spritezeiger zu retten, da diese Zeiger in Zukunft am Ende der Farbseite erwartet werden, die von XBASIC an anderer Stelle verwaltet wird („farbehires''). Nun werden die Parameter eingelesen. Sie dürfen einen Farbwert von 15 nicht überschreiten. Der Farbspeicher („farbehires'') wird anschließend mit dem Wert ZF*16+HF gefüllt, was der Forderung entspricht, daß die Zeichenfarbe in den Bits 4..7, und die Hintergrundfarbe in den Bits 0..3 steht. Die Spritezeiger werden dann noch in den Farbspeicher übertragen.

Sollte von der Multicolorgrafik auf die hochauflösende Grafik umgeschaltet worden sein, so muß außerdem noch der Farbspeicher für den Text restauriert werden. Dies erledigt die Routine <farben>, die später besprochen wird.

```
                   ;------------------------------------------------
                   ;--- XBASIC-Befehl: GRAFON ZF,HF               ---
                   ;------------------------------------------------
a974 a9 28         BBgrafon:  lda #%00101000     ;Hires Speicher festlegen
a976 20 27 aa                 jsr savesprite     ;Spritezeiger retten
a979 8d 18 d0                 sta vic+24
a97c a9 3b                    lda #%00111011     ;Hires einschalten
a97e 8d 11 d0                 sta vic+17
a981 2c f8 03                 bit colmode        ;Farbmodus ?
a984 10 03                    bpl grafon         ;nein
a986 20 5d aa                 jsr farben         ;Farbspeicher restaurieren
a989 20 55 99      grafon:    jsr getbyte        ;Zeichenfarbe holen
a98c e0 10                    cpx #16            ;gueltig ?
a98e b0 2e                    bcs nocol2         ;nein, zu gross
a990 86 fc                    stx azg+1          ;merken
a992 20 45 99                 jsr chkkom         ;auf Komma testen
a995 20 55 99                 jsr getbyte        ;Hintergrundfarbe holen
a998 e0 10                    cpx #16            ;gueltig ?
a99a 8a                       txa
a99b b0 21                    bcs nocol2         ;nein, zu gross
a99d 06 fc                    asl azg+1          ;Zeichenfarbe * 16
a99f 06 fc                    asl azg+1
a9a1 06 fc                    asl azg+1
a9a3 06 fc                    asl azg+1
a9a5 05 fc                    ora azg+1          ;+Hintergrundfarbe
a9a7 a0 c8                    ldy #>farbehires   ;azg auf Farbram setzen
```

Listing 4/6.4.4.7 (Teil 1)

```
a9a9 84 fc                   sty azg+1
a9ab a2 04                   ldx #4           ;4 Seiten setzen
a9ad a0 00                   ldy #0
a9af 84 fb                   sty azg
a9b1 91 fb       setcol:     sta (azg),y      ;eine Seite setzen
a9b3 c8                      iny
a9b4 d0 fb                   bne setcol
a9b6 e6 fc                   inc azg+1        ;bis alle Seiten gesetzt
a9b8 ca                      dex
a9b9 d0 f6                   bne setcol
a9bb 4c 40 aa                jmp loadsprite   ;Spritezeiger wieder setzen

a9be 4c 28 a9    nocol2:     jmp nocol
```

Listing 4/6.4.4.7 (Teil 2)

4/6.4.4.8

MULTION HF,F1,F2,F3

Um die Multicolorgrafik einzuschalten, benötigt man den Befehl MULTION. Mit ihm werden zusätzlich die vier möglichen Farben festgelegt, wobei die erste Farbe der Hintergrundfarbe entspricht. Bei der Multicolorgrafik wird der normale Farbspeicher für die dritte Farbe verwendet. Um beim zurückschalten in den Textmodus die Farben zu erhalten, muß der Farbspeicher in einen anderen Bereich gerettet werden. Dies übernimmt ebenfalls die Routine <farben>.

Der Befehl rettet also zunächst die Farben und die Spritezeiger. Dann wird die erste Farbe eingelesen und als Hintergrundfarbe gesetzt. Da die nächsten zwei Farben genau wie bei der hochauflösenden Grafik zu setzen sind, wird diese Routine von MULTION benutzt („JSR GRAFON"). Anschließend bleibt nur noch der dritte Farbwert in den Farbspeicher zu schreiben.

Der VIC kann dann auf die Multicolorgrafik umprogrammiert werden.

```
                  ;---------------------------------------
                  ;--- XBASIC-Befehl:MULTION HF,F1,F2,F3 -
                  ;---------------------------------------
a9c1 2c f8 03     BBmultion: bit colmode        ;Farbmodus
a9c4 30 03                   bmi nfm3           ;ja
a9c6 20 5d aa                jsr farben         ;Farbspeicher retten
a9c9 20 55 99     nfm3:      jsr getbyte        ;Hintergrundfarbe holen
a9cc 20 27 aa                jsr savesprite     ;Spritezeiger retten
a9cf e0 10                   cpx #16            ;gueltig ?
a9d1 b0 eb                   bcs nocol2         ;nein
a9d3 8e 21 d0                stx vic+33         ;Farbe setzen
a9d6 20 45 99                jsr chkkom         ;Farbe 1 & 2 setzen
a9d9 20 89 a9                jsr grafon
a9dc 20 45 99                jsr chkkom         ;Farbe 3 holen
a9df 20 55 99                jsr getbyte
a9e2 e0 10                   cpx #16            ;gueltig ?
a9e4 b0 d8                   bcs nocol2         ;nein
a9e6 a0 00                   ldy #0             ;in Farbspeicher setzen
a9e8 8a                      txa
a9e9 99 00 d8     setcol3:   sta farbram,y      ;Byte 0..255 setzen
a9ec 99 00 d9                sta 256+farbram,y  ;Byte 256..511 setzen
a9ef 99 00 da                sta 256*2+farbram,y;Byte 512..767 setzen
a9f2 99 00 db                sta 256*3+farbram,y;Byte 768..1023 setzen
a9f5 c8                      iny
a9f6 d0 f1                   bne setcol3
a9f8 a9 28                   lda #%00101000     ;Hires Speicher festlegen
a9fa 8d 18 d0                sta vic+24
a9fd a9 3b                   lda #%00111011     ;Hires einschalten
a9ff 8d 11 d0                sta vic+17
aa02 a9 18                   lda #%11000        ;Farbmodus festlegen
aa04 8d 16 d0                sta vic+22
aa07 60                      rts
```

Listing 4/6.4.4.8

4/6.4.4.9

GRAFOFF

Mit dem Befehl GRAFOFF wird die Grafik wieder abgeschaltet, egal, ob Sie die Multicolorgrafik oder die hochauflösende Grafik eingeschaltet hatten. GRAFOFF muß ggf. die Farben restaurieren, was wieder von der Routine <farben> erledigt wird. Dann wird der VIC wieder auf die normale Textdarstellung umprogrammiert, und die Flagge für die Farbgrafik gelöscht. Die geretteten Spritezeiger werden wieder in die Textseite eingesetzt.

Das Label „dografoff" bildet den Einsprung von der Eingabe-Warteschleife, um bei Fehlern oder am Programmende automatisch wieder auf die Textseite umzuschalten. Es ist hier zwar mit „BBgrafoff" identisch, hat aber ein eigenes Symbol bekommen, um während der Entwicklungsarbeit diese Automatik leicht außer Kraft setzen zu können. Wenn sie also interaktiv Grafikbefehle testen wollen, setzen Sie einfach ein RTS hinter das Symbol.

```
                 ;----------------------------------------
                 ;--- XBASIC-Befehl: GRAFOFF           ---
                 ;----------------------------------------
                 dografoff:                         ;GRAFOFF bei Fehler oder END

aa08 2c f8 03    BBgrafoff: bit colmode             ;Farbmodus
aa0b 10 03                  bpl nfm2                ;nein
aa0d 20 5d aa               jsr farben              ;Farbspeicher ruecksetzen
aa10 a9 00       nfm2:      lda #0                  ;Farbmodus abschalten
aa12 8d f8 03               sta colmode
aa15 a9 36                  lda #%00110110          ;Normale Speicherbelegung
aa17 8d 18 d0               sta vic+24
aa1a a9 1b                  lda #%00011011          ;Text einschalten
aa1c 8d 11 d0               sta vic+17
aa1f a9 08                  lda #%1000
aa21 8d 16 d0               sta vic+22
aa24 4c 40 aa               jmp loadsprite          ;Spritezeiger ruecksetzen
```

Listing 4/6.4.4.9 (Teil 1)

Retten der Spritezeiger

Die Routinen <savesprite> und <loadsprite> dienen dem Retten und Setzen eventuell vorhandener Spritezeiger. Von <savesprite> werden die Spritezeiger in den 8 Byte langen Bereich ab „spritezeig" gerettet. Dies geschieht jedoch nur, wenn sich der VIC noch in der Textdarstellung befindet, da sonst durch mehrfache Aufrufe von GRAFON die Information verlorenginge.

Mit <loadsprite> werden die Spritezeiger wieder zurückgesetzt, und zwar sowohl in die Textseite, als auch in die Farbseite für Grafik. Die Spritezeiger bleiben also in Textdarstellung und im Grafikmodus gültig.

```
                ;-----------------------------------
                ;- Spritezeiger retten/setzen      --
                ;-----------------------------------
aa27 48         savesprite:pha               ;A/Y retten
aa28 98                    tya
aa29 48                    pha
aa2a ad 18 d0              lda vic+24        ;Grafik an ?
aa2d c9 36                 cmp #%00110110
aa2f f0 0b                 beq unsavesp      ;ja
aa31 a0 07                 ldy #7            ;8 Zeiger aus textseite
aa33 b9 f8 cf   savespd:   lda textseite+1024-8,y;speichern
aa36 99 55 aa              sta spritezeig,y
aa39 88                    dey               ;bis alle Zeiger gespeichert
aa3a 10 f7                 bpl savespd
aa3c 68         unsavesp:  pla               ;A/Y restaurieren
aa3d a8                    tay
aa3e 68                    pla
aa3f 60                    rts

aa40 48         loadsprite:pha               ;A/Y retten
aa41 98                    tya
aa42 48                    pha
aa43 a0 07                 ldy #7            ;8 Spritezeiger aus Puffer in
aa45 b9 55 aa   loadspd:   lda spritezeig,y  ;textseite und
aa48 99 f8 cf              sta textseite+1024-8,y;farbehires bringen
aa4b 99 f8 cb              sta farbehires+1024-8,y
aa4e 88                    dey
aa4f 10 f4                 bpl loadspd
aa51 68                    pla               ;A/Y restaurieren
aa52 a8                    tay
aa53 68                    pla
aa54 60                    rts

                spritezeig:.db 8             ;Platz fuer 8 Spritezeiger
```

Listing 4/6.4.4.9 (Teil 2)

Retten der Farbinformation

Die Routine <farben> wurde schon häufig angesprochen. Sie soll sowohl den Farbspeicher in den Bereich „farbmem" retten, als auch die Farben aus diesem zurückholen. Dies wird erreicht, indem die beiden Inhalte miteinander vertauscht werden. Dabei kann man ausnutzen, daß der Farbspeicher des C 64 aus einem zusätzlichen Speicherbereich von einem KByte mit nur 4 Bits besteht. Man kann den gesamten Farbbereich also in einem 512 Byte umfassenden Puffer retten, indem man jeweils zwei Farbinformationen zu je 4 Bit in einem Byte zusammenfaßt. Genau dies wird hier getan.

Zusätzlich wird noch die Hintergrundfarbe auf die gleiche Weise gerettet bzw. zurückgeholt, da diese bei der Multicolorgrafik verändert wird.

```
                  ;-------------------------------------
                  ;- Farbram in/aus Puffer retten/holen --
                  ;-------------------------------------
aa5d a9 00        farben:    lda #<farbram      ;azg auf Farbram
aa5f 85 fb                   sta azg
aa61 a9 d8                   lda #>farbram
aa63 85 fc                   sta azg+1
aa65 a9 00                   lda #<colmem       ;azg2 auf colmem
aa67 85 fd                   sta azg2
aa69 a9 06                   lda #>colmem
aa6b 85 fe                   sta azg2+1
aa6d a0 00        transcols: ldy #0             ;Zwei Nibble aus Farbram holen
aa6f b1 fd                   lda (azg2),y       ;Byte in colmem auf Stack
aa71 48                      pha
aa72 b1 fb                   lda (azg),y        ;erstes Nibble in Bit 4..7
aa74 0a                      asl
aa75 0a                      asl
aa76 0a                      asl
aa77 0a                      asl
aa78 8d 3c 03                sta mem            ;merken
aa7b c8                      iny
aa7c b1 fb                   lda (azg),y        ;zweites Nibble in Bit 0..3
aa7e 29 0f                   and #%1111
aa80 0d 3c 03                ora mem            ;zweites Nibble einblenden
aa83 88                      dey
aa84 91 fd                   sta (azg2),y       ;in colmem speichern
aa86 68                      pla                ;Byte aus colmem in Accu
aa87 48                      pha
aa88 4a                      lsr                ;Bit 4..7 in Farbram speichern
aa89 4a                      lsr
aa8a 4a                      lsr
aa8b 4a                      lsr
aa8c 91 fb                   sta (azg),y
aa8e c8                      iny                ;Bit 0..3 in Farbram speichern
aa8f 68                      pla
aa90 91 fb                   sta (azg),y
aa92 e6 fd                   inc azg2           ;Zeiger auf colmem +1
aa94 d0 02                   bne cc
aa96 e6 fe                   inc azg2+1
aa98 18           cc:        clc                ;Zeiger auf Farbram +2
aa99 a5 fb                   lda azg
aa9b 69 02                   adc #2
aa9d 85 fb                   sta azg
aa9f 90 02                   bcc cc2
aaa1 e6 fc                   inc azg+1
aaa3 a5 fc        cc2:       lda azg+1          ;Ende des Farbspeichers erreicht ?
aaa5 c9 dc                   cmp #>farbram+1024
aaa7 d0 c4                   bne transcols      ;nein
aaa9 ad f8 03                lda colmode        ;Marke fuer Farbmodus umdrehen
aaac 49 80                   eor #128
aaae 8d f8 03                sta colmode
aab1 ad c0 aa                lda hcol           ;Hintergrundfarbe in hcol speichern
aab4 48                      pha
aab5 ad 21 d0                lda vic+33         ;und hcol als Hintergrundfarbe setzen
aab8 8d c0 aa                sta hcol
aabb 68                      pla
aabc 8d 21 d0                sta vic+33
aabf 60                      rts

aac0 00           hcol:      .by 0              ;Speicher fuer Hintergrundfarbe
```

Listing 4/6.4.4.9 (Teil 3)

4/6.4.4.10

MOVE X,Y

Der Befehl MOVE dient dem Setzen des Grafikcursors. Da dies dem Einlesen eines Koordinatenpaares entspricht, wie es von vielen Befehlen benötigt wird, wird die Routine zusätzlich mit dem Symbol „getgkoor" versehen.

Die Routine holt sich das Koordinatenpaar als Adresse und Byte, wie es z.B. der Befehl ROKE tut. Dieses Paar wird auf den zulässigen Bereich überprüft, d.h. die Y-Koordinate darf den Wert 199 nicht überschreiten, und die X-Koordinate den Wert 319 in hochauflösender Grafik, bzw. den Wert 159 in Multicolorgrafik nicht überschreiten.

Ist das Koordinatenpaar gültig, wird es gespeichert.

```
             ;----------------------------------------
             ;--- Koordinatenpaar einlesen          ---
             ;--- & XBASIC-Befehl: MOVE X,Y         ---
             ;----------------------------------------
aac1 a5 15   koor160:   lda $15          ;X > 159 ?
aac3 d0 71              bne nocol3       ;ja, Fehler
aac5 a5 14              lda $14
aac7 c9 a0              cmp #160
aac9 b0 6b              bcs nocol3       ;ja, Fehler
aacb 90 1a              bcc xok          ;nein, X ok

             BBmove:
aacd 20 68 99 getgkoor: jsr getadby      ;X und Y holen
aad0 e0 c8   setgkoor:  cpx #200         ;Y <200 ?
aad2 b0 62              bcs nocol3       ;nein, Y zu gross
aad4 2c f8 03           bit colmode      ;Farbmodus ?
aad7 30 e8              bmi koor160      ;ja
aad9 a5 15              lda $15          ;X < 512 ?
aadb f0 0a              beq xok          ;(<256 passt immer)
aadd c9 02              cmp #2
aadf b0 55              bcs nocol3       ;nein
aae1 a5 14              lda $14
aae3 c9 40              cmp #<320        ;< 320 ?
aae5 b0 4f              bcs nocol3       ;nein
aae7 8e fc 03 xok:      stx ycurs        ;Koordinaten speichern
aaea a5 14              lda $14
aaec 8d fa 03           sta xcurs
aaef a5 15              lda $15
aaf1 8d fb 03           sta xcurs+1
aaf4 60                 rts
```

Listing 4/6.4.4.10

4/6.4.4.11

MOVER X,Y

Manchmal ist es aber auch nützlich, die Koordinaten relativ zu den letzten Koordinaten anzugeben. Dies ermöglicht die Routine <getrgkoor>, die dem MOVER-Befehl entspricht. Die Routine holt zwei Integerzahlen, und addiert diese zu den bisherigen Koordinaten. Die Y-Koordinate wird dabei bereits auf den zulässigen Bereich überprüft. Das Überprüfen der X-Koordinate wird von der Routine für das absolute Setzen übernommen, welche die restliche Arbeit verrichtet.

```
                  BBmover:                        ;relative Koordinaten holen
aaf5 20 2c ab     getrgkoor: jsr getinteger       ;Integer holen
aaf8 18                      clc                  ;Wert + X-Koordinate
aaf9 98                      tya                  ;Low
aafa 6d fa 03                adc xcurs
aafd 8d 3c 03                sta mem              ;merken
ab00 8a                      txa                  ;High
ab01 6d fb 03                adc xcurs+1
ab04 8d 3d 03                sta mem+1            ;merken
ab07 20 45 99                jsr chkkom           ;Integer holen
ab0a 20 2c ab                jsr getinteger
ab0d 18                      clc                  ;+ Y Koordinate
ab0e 98                      tya
ab0f 6d fc 03                adc ycurs
ab12 8d 3e 03                sta mem+2            ;speichern
ab15 8a                      txa
ab16 69 00                   adc #0
ab18 c9 00                   cmp #0
ab1a d0 1a                   bne nocol3           ;Y>199 -> Fehler
ab1c ae 3e 03                ldx mem+2            ;Weiter bei den absoluten
ab1f ad 3c 03                lda mem              ;Koordinaten
ab22 85 14                   sta $14
ab24 ad 3d 03                lda mem+1
ab27 85 15                   sta $15
ab29 4c d0 aa                jmp setgkoor

ab2c 20 2d 99     getinteger:jsr frmevl           ;arithm. Ausdruck holen
ab2f a2 3c                   ldx #60              ;in Integer wandeln
ab31 20 d8 98                jsr basicrom         ;Integer in Y/X
ab34 aa                      tax
ab35 60                      rts

ab36 4c 28 a9     nocol3:    jmp nocol
```

Listing 4/6.4.4.11

4/6.4.4.12

A=TEST(X,Y) / A=TESTR(X,Y)

Die Funktion TEST liefert die Farbe des Punktes mit den Koordinaten (X,Y) zurück. Je nachdem, ob TEST oder TESTR aufgerufen wird, werden die Koordinaten absolut oder relativ angegeben. Die Befehle unterscheiden sich nur in der Verwendung der Routine <getgkoor> bzw. <getrgkoor>.

In der hochauflösenden Grafik ist der Befehl sehr einfach: die Routine <makezg> wird aufgerufen. Sie liefert in „azg" einen Zeiger auf das Byte, in dem der Punkt liegt. In „azg2+1" liefert sie den Wert unter dieser Adresse, und in „azg2" die Maske, in der genau das Bit gesetzt ist, das dem Punkt entspricht. Durch AND-Verknüpfung von „azg2" und „azg2+1" kann so festgestellt werden, ob der Punkt gesetzt ist. Je nach Ergebnis wird dann Null oder Eins als Ergebnis geliefert.

In der Multicolorgrafik kann das Ergebnis einen Wert zwischen Null und drei haben. Die Routine <makezg> liefert wieder die bekannten Daten. Zusätzlich steht nun im X-Register die Nummer der Maske. Durch Division durch zwei erhält man den Offset für die bei PEN definierte Tabelle. Nun wird der Bytewert (in „azg+1") solange um zwei Bit nach rechts geschoben, bis der Farbwert des Punktes in den unteren zwei Bit steht. Die restlichen Bits werden gelöscht, und der so erhaltene Wert zurückgeliefert.

```
                ;-------------------------------------
                ;--- XBASIC-Befehl: A=TEST(X,Y)     ---
                ;-------------------------------------
ab39 20 73 00   BBtestr:   jsr chrget       ;relative Koordinaten holen
ab3c 20 f5 aa              jsr getrgkoor
ab3f 4c 48 ab              jmp testbef      ;und testen

ab42 20 73 00   BBtest:    jsr chrget       ;Koordinaten holen
ab45 20 cd aa              jsr getgkoor
ab48 20 35 99   testbef:   jsr chkklazu     ;auf ')' testen
ab4b 20 aa ab              jsr makezg       ;Pointer berechnen
ab4e 2c f8 03              bit colmode      ;Farbmodus ?
ab51 10 15                 bpl hirestst     ;nein
ab53 8a                    txa              ;Maskennummer in A
ab54 4a                    lsr              ;/2
ab55 aa                    tax              ;in X
ab56 a5 fe                 lda azg2+1       ;Wert von (Pointer)
ab58 ca         schiebe:   dex              ;rechtsschieben
ab59 30 05                 bmi schiebeend   ;bis signifikante Bits
```

Listing 4/6.4.4.12 (Teil 1)

```
ab5b 4a                      lsr                ;in Bit 0/1
ab5c 4a                      lsr
ab5d 4c 58 ab                jmp schiebe
ab60 29 03      schiebeend:and #%11             ;andere Bits ausblenden
ab62 a8                      tay                ;A/Y ist Ergebnis
ab63 a9 00                   lda #0
ab65 4c 53 a1                jmp ergebnis

ab68 a5 fe      hireststs:   lda azg2+1         ;Wert von (Pointer)
ab6a 25 fd                   and azg2           ;AND mit Maske
ab6c d0 03                   bne coleins        ;Farbe 1
ab6e a0 00                   ldy #0             ;0 liefern
ab70 2c                      bit
ab71 a0 01      coleins:     ldy #1             ;1 liefern
ab73 a9 00                   lda #0
ab75 4c 53 a1                jmp ergebnis
```

Listing 4/6.4.4.12 (Teil 2)

4/6.4.4.13

PLOT X,Y / PLOTR X,Y

Mit den Befehlen PLOT und PLOTR kann ein Punkt auf dem Bildschirm gesetzt werden. Die Befehle unterscheiden sich wieder nur durch die Verwendung der Routine <getgkoor> bzw. <getrgkoor>. Die Routine <makezg> liefert wieder die oben beschriebenen Daten.

In der Multicolorgrafik werden die dem Punkt entsprechenden Bits ausgeblendet, und die Farbbits per OR eingefügt. Dies ist relativ einfach, weil man mit Hilfe des Wertes im X-Register auf die bei PEN definierten Tabellen zugreifen kann.

In der Hiresgrafik wird zum Setzen des Punktes die Werte in „azg2" und „azg+1" OR-verknüpft. Zum Löschen des Punktes wird die Maske erst inveriert, und dann mit dem Wert in „azg2" AND-verknüpft.

```
                    ;-----------------------------------------
                    ;--- XBASIC-Befehl: PLOT X,Y          ---
                    ;-----------------------------------------
ab78 20 f5 aa       BBplotr:  jsr getrgkoor     ;relative Koordinaten lesen
ab7b 4c 81 ab                 jmp plot          ;Punkt setzen

ab7e 20 cd aa       BBplot:   jsr getgkoor      ;Punkt setzen
ab81 20 aa ab       plot:     jsr makezg        ;Adresse in azg bringen / Maske in az
ab84 2c f8 03       doplot:   bit colmode       ;Farbmodus ?
ab87 10 0e                    bpl hires         ;nein
ab89 8a                       txa               ;Maskennummer/2 in X
ab8a 4a                       lsr
ab8b aa                       tax
ab8c a5 fe                    lda azg2+1        ;Wert von (Pointer)
ab8e 3d 47 a9                 and delbits,x     ;Bits loeschen
ab91 1d 43 a9                 ora penbits,x     ;PEN-Bits einblenden
ab94 91 fb                    sta (azg),y       ;Punkt setzen
ab96 60                       rts

ab97 a5 fd          hires:    lda azg2
ab99 2c f9 03                 bit drawmode      ;Drawmode ist setzen ?
ab9c 30 05                    bmi delmode       ;nein
ab9e 05 fe                    ora azg2+1        ;Bit setzen
aba0 91 fb                    sta (azg),y       ;Punkt setzen
aba2 60                       rts
aba3 49 ff          delmode:  eor #$ff          ;Maske invertieren
aba5 25 fe                    and azg2+1        ;Bit loeschen
aba7 91 fb                    sta (azg),y       ;Punkt loeschen
aba9 60                       rts
```

Listing 4/6.4.4.13 (Teil 1)

Die Routine MAKEZG

Hier folgt nun die bereits vorgestellte Routine <makezg>. Sie berechnet den Zeiger in „azg" und die Maske in „azg2" nach den Formeln:

```
azg := hiresseite+320*int(Y/8)+(Y and 7)+8*int(x/8)
azg2:=255-2^(7(x and 7))
```

Dann wird der Wert von (azg) in „azg2+1" abgespeichert. Die Maskennummer in X entsteht dabei zwangsläufig, da die Maske mit Hilfe des X-Registers aus einer Tabelle ausgelesen wird.

```
                ;------------------------------------
                ;--- Berechnet aus Koordinaten Zeiger ---
                ;--- auf Grafikbyte in azg, Bitmaske  ---
                ;--- in azg2 und Bytewert in azg2+1   ---
                ;------------------------------------
abaa 2c f8 03   makezg:   bit colmode        ;Farbmodus ?
abad 10 10                bpl makezg1        ;nein
abaf 0e fa 03             asl xcurs          ;X-Koordinate *2
abb2 2e fb 03             rol xcurs+1
abb5 20 bf ab             jsr makezg1        ;Zeiger berechnen
abb8 4e fb 03             lsr xcurs+1        ;X-Koordinate /2
abbb 6e fa 03             ror xcurs
abbe 60                   rts

abbf ac fc 03   makezg1:  ldy ycurs          ;INT(Y/8)
abc2 98                   tya
abc3 4a                   lsr
abc4 4a                   lsr
abc5 4a                   lsr
abc6 aa                   tax
abc7 bd 16 ac             lda mulhi,x
abca 85 fc                sta azg+1          ;>320*INT(Y/8)
abcc 8a                   txa
abcd 29 03                and #3             ;Bits 2..7 loeschen
abcf aa                   tax
abd0 bd 30 ac             lda mullo,x
abd3 85 fb                sta azg            ;<320*INT(Y/8)
abd5 98                   tya                ;Y Koordinate
abd6 29 07                and #7             ;AND 7
abd8 18                   clc
abd9 65 fb                adc azg            ;Offset Y = 320*INT(Y/8)+(Y AND 7)
abdb 85 fb                sta azg
abdd ad fa 03             lda xcurs
abe0 29 f8                and #$f8
abe2 85 fe                sta azg2+1         ;Offset X = 8*INT(X/8)
abe4 a9 e0                lda #>hiresseite   ;Hiresseite
abe6 05 fc                ora azg+1
abe8 85 fc                sta azg+1
abea 18                   clc                ;Adresse=Seite +Offset X +Offset Y
abeb a5 fb                lda azg
abed 65 fe                adc azg2+1
abef 85 fb                sta azg
abf1 a5 fc                lda azg+1
abf3 6d fb 03             adc xcurs+1
abf6 85 fc                sta azg+1
abf8 ad fa 03             lda xcurs
abfb 29 07                and #7
abfd 49 07                eor #7             ;7-(X AND 7)
abff aa                   tax
ac00 bd 34 ac             lda masken,x       ;Maske holen
ac03 85 fd                sta azg2           ;und speichern
ac05 78                   sei
ac06 a9 34                lda #$34           ;auf RAM umschalten
ac08 85 01                sta pp
ac0a a0 00                ldy #0             ;Wert von (Pointer)
ac0c b1 fb                lda (azg),y
ac0e 85 fe                sta azg2+1         ;merken
ac10 a9 36                lda #$36           ;wieder normale Speicherverteilung
ac12 85 01                sta pp
ac14 58                   cli
ac15 60                   rts

                ;Tabelle x*320
```

Listing 4/6.4.4.13 (Teil 2-1)

```
ac16 00 01 02 473   mulhi:     .by 0,1,2,3,5,6,7,8,10,11,12,13,15
03 05 06 07 08 0a 0b 0c 0d 0f
ac23 10 11 12 474              .by 16,17,18,20,21,22,23,25,26,27,28,30,31
14 15 16 17 19 1a 1b 1c 1e 1f
              475
ac30 00 40 80 476   mullo:     .by 0,$40,$80,$c0
c0
              477
              478   ;Tabelle der Masken
ac34 01 02 04 479   masken:    .by 1,2,4,8,$10,$20,$40,$80
08 10 20 40 80
              480
```

Listing 4/6.4.4.13 (Teil 2-2)

4/6.4.4.14

LINE X,Y / LINER X,Y

Der LINE-Befehl zieht eine Linie von der letzten Grafikcursorposition zu der angegebenen Position. Bei dem LINER-Befehl werden die Koordinaten wieder relativ zu den letzten Koordinaten angegeben.

Der LINE-Algorithmus ist nicht so einfach zu durchschauen. Dennoch soll er hier nicht weiter analysiert werden, da der LINE-Algorithmus bereits im Beitrag zur 3D-Grafik behandelt wurde, und auch in einem Grundlagenbeitrag vorgestellt werden wird.

Im Prinzip wird der erste Punkt mit PLOT gesetzt, und dann durch vertikale und horizontale Schritte zum zweiten Punkt gewandert.

Für das Verständnis der weiteren Routinen ist nur wichtig zu wissen, daß eine Linie zwischen den Koordinaten (xcurs,ycurs) und (xcurs2,ycurs2) gezogen wird, und zwar in beiden Auflösungen.

```
                  ;-------------------------------------------
                  ;--- XBASIC-Befehl: LINE X,Y              ---
                  ;-------------------------------------------
                  linemem:   .eq 87          ;temporaerer Zwischenspeicher fuer LINE

ac3c ad fa 03     BBliner:   lda xcurs       ;alten Grafikcursor retten
ac3f 8d fd 03                sta xcurs2
ac42 ad fb 03                lda xcurs+1
ac45 8d fe 03                sta xcurs2+1
ac48 ad fc 03                lda ycurs
ac4b 8d ff 03                sta ycurs2
ac4e 20 f5 aa                jsr getrgkoor   ;Zielkoordinaten relativ lesen
ac51 4c 69 ac                jmp line

ac54 ad fa 03     BBline:    lda xcurs       ;alten Grafikcursor retten
ac57 8d fd 03                sta xcurs2
ac5a ad fb 03                lda xcurs+1
ac5d 8d fe 03                sta xcurs2+1
ac60 ad fc 03                lda ycurs
ac63 8d ff 03                sta ycurs2
ac66 20 cd aa                jsr getgkoor    ;Zielkoordinaten absolut lesen

ac69 20 81 ab     line:      jsr plot        ;Punkt setzen
ac6c 86 5e                   stx linemem+7   ;Maskennummer merken

ac6e 78           coloff1:   sei             ;auf RAM schalten
ac6f a9 34                   lda #$34
ac71 85 01                   sta pp
ac73 ad fd 03                lda xcurs2      ;X2-X1
ac76 38                      sec
ac77 ed fa 03                sbc xcurs
ac7a 48                      pha
ac7b ae fe 03                ldx xcurs2+1
ac7e 8a                      txa
ac7f ed fb 03                sbc xcurs+1
ac82 85 5a                   sta linemem+3   ;speichern
ac84 b0 0a                   bcs vorzw       ;Ergebnis negativ ?
ac86 68                      pla             ;ja
ac87 49 ff                   eor #$ff
ac89 69 01                   adc #1
ac8b 48                      pha
ac8c a9 00                   lda #0
ac8e e5 5a                   sbc linemem+3   ;Vorzeichen wechseln
ac90 85 58        vorzw:     sta linemem+1   ;>X2-X1
ac92 85 5c                   sta linemem+5
ac94 68                      pla
ac95 85 57                   sta linemem     ;<X2-X1
ac97 85 5b                   sta linemem+4
ac99 ad ff 03                lda ycurs2
ac9c 18                      clc             ;Y2-Y1
ac9d ed fc 03                sbc ycurs       ;Ergebnis negativ ?
aca0 90 04                   bcc negat
aca2 49 ff                   eor #$ff        ;nein
aca4 69 fe                   adc #$fe        ;Vorzeichenwechsel
aca6 85 59        negat:     sta linemem+2   ;Y2-Y1
aca8 66 5a                   ror linemem+3   ;(X2-X1)/2
acaa 38                      sec
acab e5 57                   sbc linemem     ;(Y2-Y1)-(X2-X1)
acad aa                      tax             ;Low Byte in X
acae a9 ff                   lda #$ff
acb0 e5 58                   sbc linemem+1
acb2 85 5d                   sta linemem+6   ;High Byte
acb4 b0 11                   bcs schleife    ;unbedingter Sprung
acb6 2c f8 03     hstep:     bit colmode     ;Farbmodus
```

Listing 4/6.4.4.14 (Teil 1)

```
acb9 30 07              bmi colan         ;ja
acbb 0a                 asl               ;horizontaler Schritt
acbc 20 59 ad           jsr horizontal
acbf 38                 sec
acc0 b0 05              bcs schleife
acc2 0a        colan:   asl               ;horizontaler Schritt in Farbe
acc3 20 3f ad           jsr horizcol
acc6 38                 sec
acc7 a5 5b     schleife: lda linemem+4
acc9 65 59              adc linemem+2
accb 85 5b              sta linemem+4
accd a5 5c              lda linemem+5
accf e9 00              sbc #0            ;(X2-X1)-(Y2-Y1) nach (X2-X1)
acd1 85 5c     setzen:  sta linemem+5
acd3 08                 php
acd4 a0 00              ldy #0
acd6 b1 fb              lda (azg),y       ;Wert von (Pointer) lesen
acd8 85 fe              sta azg2+1
acda 86 5f              stx linemem+8     ;X merken
acdc a5 5e              lda linemem+7     ;Maske *2 in X
acde 0a                 asl
acdf aa                 tax
ace0 20 84 ab           jsr doplot        ;Punkt setzen
ace3 a6 5f              ldx linemem+8     ;alten X-Wert lesen
ace5 28                 plp
ace6 e8                 inx               ;Weiter bis X=0
ace7 d0 0a              bne lcont
ace9 e6 5d              inc linemem+6     ;weiter bis Zaehler =0
aceb d0 06              bne lcont
aced a9 36              lda #$36          ;Speicher wieder normal
acef 85 01              sta pp
acf1 58                 cli
acf2 60        coloff2: rts               ;fertig

acf3 a5 5a     lcont:   lda linemem+3
acf5 b0 bf              bcs hstep
acf7 20 07 ad           jsr vertikal      ;vertikaler Schritt
acfa 18                 clc
acfb a5 5b              lda linemem+4
acfd 65 57              adc linemem
acff 85 5b              sta linemem+4
ad01 a5 5c              lda linemem+5
ad03 65 58              adc linemem+1
ad05 50 ca              bvc setzen        ;unbedingter Sprung

               ;Vertikaler Schritt
ad07 30 1a     vertikal: bmi unten        ;Schritt nach unten
ad09 a5 fb              lda azg           ;Schritt nach oben
ad0b 29 07              and #7
ad0d f0 05              beq oben1         ;oberer Blockrand
ad0f 18                 clc
ad10 a9 ff              lda #$ff
ad12 90 04              bcc oben2         ;-1
ad14 a9 c7     oben1:   lda #$c7          ;-327
ad16 c6 fc              dec azg+1
ad18 65 fb     oben2:   adc azg
ad1a 85 fb              sta azg
ad1c a5 fc              lda azg+1
ad1e e9 00              sbc #0
ad20 85 fc              sta azg+1
ad22 60                 rts

ad23 a5 fb     unten:   lda azg           ;Schritt nach unten
ad25 29 07              and #7            ;Blockrand unten
```

Listing 4/6.4.4.14 (Teil 2)

```
ad27 c9 07              cmp #7
ad29 f0 05              beq unten1
ad2b 38                 sec
ad2c a9 00              lda #0
ad2e b0 04              bcs unten2          ;+1
ad30 a9 38    unten1:   lda #$38            ;+313
ad32 e6 fc              inc azg+1
ad34 65 fb    unten2:   adc azg
ad36 85 fb              sta azg
ad38 a9 00              lda #0
ad3a 65 fc              adc azg+1
ad3c 85 fc              sta azg+1
ad3e 60                 rts

ad3f 10 3e    horizcol: bpl colrechts
ad41 e6 5e              inc linemem+7       ;Schritt nach links (Farbe)
ad43 a5 5e              lda linemem+7       ;Maskennummer+1
ad45 c9 04              cmp #4              ;<4 ?
ad47 90 23              bcc links1          ;dann fertig
ad49 a9 00              lda #0              ;Maskennummer =0
ad4b 85 5e              sta linemem+7
ad4d a5 fb              lda azg             ;Pointer-8
ad4f 38                 sec
ad50 e9 08              sbc #8
ad52 85 fb              sta azg
ad54 b0 16              bcs links1
ad56 c6 fc              dec azg+1
ad58 60                 rts

ad59 10 12    horizontal:bpl rechts
ad5b 06 fd              asl azg2            ;Schritt nach links
ad5d 90 0d              bcc links1          ;Maske schieben
ad5f 26 fd              rol azg2            ;Byte ueberschritten,
ad61 a5 fb              lda azg             ;Pointer-8
ad63 38                 sec
ad64 e9 08              sbc #8
ad66 85 fb              sta azg
ad68 b0 02              bcs links1
ad6a c6 fc              dec azg+1
ad6c 60       links1:   rts

ad6d 46 fd    rechts:   lsr azg2            ;Schritt nach rechts
ad6f 90 0d              bcc rechts2         ;Maske schieben
ad71 66 fd              ror azg2            ;Bereich ueberschritten,
ad73 a5 fb              lda azg             ;Pointer+8
ad75 18                 clc
ad76 69 08              adc #8
ad78 85 fb              sta azg
ad7a 90 02              bcc rechts2
ad7c e6 fc              inc azg+1
ad7e 60       rechts2:  rts

ad7f c6 5e    colrechts: dec linemem+7      ;Schritt nach rechts (Farbe)
ad81 10 fb              bpl rechts2         ;Maskennummer-1
ad83 a9 03              lda #3              ;<0, dann Maskennummer=3
ad85 85 5e              sta linemem+7       ;Pointer+8
ad87 a5 fb              lda azg
ad89 18                 clc
ad8a 69 08              adc #8
ad8c 85 fb              sta azg
ad8e 90 ee              bcc rechts2
ad90 e6 fc              inc azg+1
ad92 60                 rts
```

Listing 4/6.4.4.14 (Teil 3)

4/6.4.4.15

FRAME X1,Y1,X2,Y2

Mit dem FRAME-Befehl wird ein Rahmen zwischen den Punkten (X1,Y1) und (X2,Y2) gezeichnet. Dabei ist es egal, wo diese Punkte auf dem Bildschirm liegen, da zwischen zwei Punkten immer ein Rechteck gezeichnet werden kann.

Die eingelesenen Koordinaten werden von der Routine <inmem> hintereinander ab „mem'' gespeichert, so das sich die folgende Struktur ergibt:

	<X1	>X1	Y1	<X2	>X2	Y2
mem +	0	1	2	3	4	5

Der Rahmen kann nun durch vier Linien beschrieben werden, wobei jede Linie wiederum durch 6 Byte beschrieben wird. In einer Tabelle werden daher für die vier Linien der Offset der Koordinaten abgelegt, so das durch Abarbeiten dieser Tabelle der Rahmen gezeichnet wird.

```
                ;-----------------------------------------
                ;---- XBASIC-Befehl: FRAME X1,Y1,X2,Y2 ---
                ;-----------------------------------------
ad93 20 cd aa   BBframe:   jsr getgkoor       ;Koordinaten 1 holen
ad96 a0 00                 ldy #0             ;ab mem speichern
ad98 20 c9 ad              jsr inmem
ad9b 20 45 99              jsr chkkom         ;Koordinaten 2 holen
ad9e 20 cd aa              jsr getgkoor
ada1 a0 03                 ldy #3             ;ab mem+3 speichern
ada3 20 c9 ad              jsr inmem
ada6 a0 17                 ldy #4*6-1         ;4 Linien ziehen
ada8 a2 05      oneline:   ldx #5             ;pro Linie 6 Werte
adaa 8e 42 03              stx mem+6          ;Zaehler setzen
adad be dc ad   onelinel:  ldx permuts,y      ;Index der Werte holen
adb0 bd 3c 03              lda mem,x          ;Wert holen
adb3 ae 42 03              ldx mem+6          ;und als Linienbeschreibung in der Form
adb6 9d fa 03              sta xcurs,x        ;Xa Ya Xb Yb speichern
adb9 88                    dey                ;bis alle Werte gesetzt
adba ce 42 03              dec mem+6
adbd 10 ee                 bpl onelinel
adbf 98                    tya                ;Y retten
adc0 48                    pha
adc1 20 69 ac              jsr line           ;Line von Xa Ya nach Xb Yb ziehen
adc4 68                    pla
adc5 a8                    tay
adc6 10 e0                 bpl oneline        ;bis alle Linien gezogen
```

Listing 4/6.4.4.15 (Teil 1)

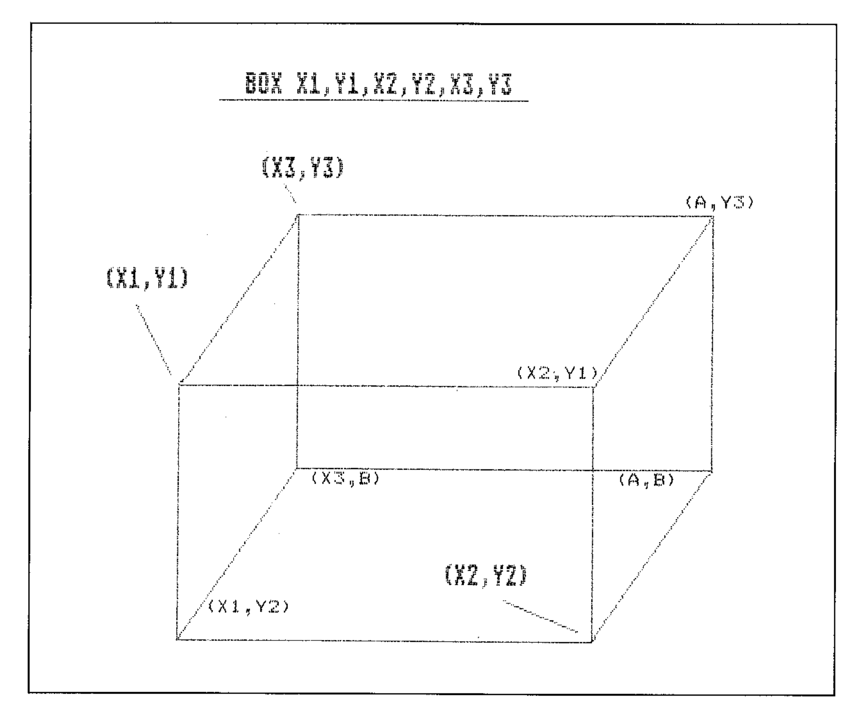
```
adc8 60          699              rts
                 700
adc9 ad fa 03 701     inmem:      lda xcurs        ;Koordinaten ab mem,y speichern
adcc 99 3c 03 702                 sta mem,y
adcf ad fb 03 703                 lda xcurs+1
add2 99 3d 03 704                 sta mem+1,y
add5 ad fc 03 705                 lda ycurs
add8 99 3e 03 706                 sta mem+2,y
addb 60          707              rts
                 708
addc 03 04 05 709     permuts:    .by 3,4,5,3,4,2  ;Indices der Werte
03 04 02
ade2 03 04 02 710                 .by 3,4,2,0,1,2  ;der Linien
00 01 02
ade8 00 01 02 711                 .by 0,1,2,0,1,5
00 01 05
adee 00 01 05 712                 .by 0,1,5,3,4,5
03 04 05
                 713
```

Listing 4/6.4.4.15 (Teil 2)

4/6.4.4.16

BOX X1,Y1,X2,Y2,X3,Y3

Der Befehl BOX stellt eine Erweiterung des Befehls FRAME dar: er zeichnet einen Quader. Die Bedeutung der Parameter können sie der folgenden Zeichnung entnehmen:



Mit A=X3+X2-X1 UND B=Y3+Y2-Y1.

Der Quader kann nach dem gleichen Schema wie der Rahmen gezeichnet werden, wobei man noch zwei Werte aus den drei Koordinatenpaaren berechnen muß, um alle Punkte beschreiben zu können (siehe Skizze). Als Anwender müssen Sie beachten, daß der Quader vollständig auf den Bildschirm passen muß. Andernfalls wird der Fehler „Illegal Quantity" gemeldet.

```
                ;-------------------------------------
                ;--- XB-Befehl: BOX X1,Y1,X2,Y2,X3,Y3 --
                ;-------------------------------------
adf4 20 cd aa   BBbox:    jsr getgkoor      ;Koordinaten 1 holen
adf7 a0 00                ldy #0            ;ab mem speichern
adf9 20 c9 ad             jsr inmem
adfc 20 45 99             jsr chkkom        ;Koordinaten 2 holen
adff 20 cd aa             jsr getgkoor
ae02 a0 03                ldy #3            ;ab mem+3 speichern
ae04 20 c9 ad             jsr inmem
ae07 20 45 99             jsr chkkom        ;Koordinaten 3 holen
ae0a 20 cd aa             jsr getgkoor
ae0d a0 06                ldy #6            ;ab mem+6 speichern
ae0f 20 c9 ad             jsr inmem
ae12 18                   clc
ae13 ad 42 03             lda mem+6         ;A=X3+X2-X1
ae16 6d 3f 03             adc mem+3
ae19 8d 45 03             sta mem+9
ae1c ad 43 03             lda mem+7
ae1f 6d 40 03             adc mem+4
ae22 8d 46 03             sta mem+10
ae25 38                   sec
ae26 ad 45 03             lda mem+9
ae29 ed 3c 03             sbc mem+0
ae2c 8d 45 03             sta mem+9
ae2f ad 46 03             lda mem+10
ae32 ed 3d 03             sbc mem+1
ae35 8d 46 03             sta mem+10
ae38 18                   clc
ae39 ad 44 03             lda mem+8         ;B=Y3+Y2-Y1
ae3c 6d 41 03             adc mem+5
ae3f aa                   tax
ae40 a9 00                lda #0
ae42 69 00                adc #0
ae44 a8                   tay
ae45 38                   sec
ae46 8a                   txa
ae47 ed 3e 03             sbc mem+2
ae4a 8d 47 03             sta mem+11
ae4d 98                   tya
ae4e e9 00                sbc #0
ae50 d0 4d                bne nocol4        ;B zu klein
ae52 ad 47 03             lda mem+11
ae55 c9 c8                cmp #200
ae57 b0 46                bcs nocol4        ;B zu gross
ae59 2c f8 03             bit colmode       ;Farbmodus ?
ae5c 30 12                bmi cxcolm        ;ja
ae5e ad 46 03             lda mem+10        ;A<320 ?
ae61 f0 19                beq box
ae63 c9 02                cmp #2
ae65 b0 38                bcs nocol4        ;nein (>511)
ae67 ad 45 03             lda mem+9
ae6a c9 40                cmp #<320         ;nein
ae6c b0 31                bcs nocol4
ae6e 90 0c                bcc box

ae70 ad 46 03   cxcolm:   lda mem+10        ;x<160 ?
ae73 d0 2a                bne nocol4        ;nein
ae75 ad 45 03             lda mem+9
```

Listing 4/6.4.4.16 (Teil 1)

```
ae78 c9 a0                    cmp #160
ae7a b0 23                    bcs nocol4          ;nein

ae7c a0 47       box:         ldy #12*6-1         ;12 Linien ziehen
ae7e a2 05       onelinebo:   ldx #5              ;a 6 Werte
ae80 8e 48 03                 stx mem+12          ;Zaehler
ae83 be a2 ae    onelinelbo:  ldx permuts2,y      ;Index holen
ae86 bd 3c 03                 lda mem,x           ;Wert holen
ae89 ae 48 03                 ldx mem+12          ;und speichern
ae8c 9d fa 03                 sta xcurs,x
ae8f 88                       dey                 ;bis 6 Werte gesetzt
ae90 ce 48 03                 dec mem+12
ae93 10 ee                    bpl onelinelbo
ae95 98                       tya                 ;Linie ziehen
ae96 48                       pha
ae97 20 69 ac                 jsr line
ae9a 68                       pla
ae9b a8                       tay
ae9c 10 e0                    bpl onelinebo       ;weiter bis 12 Linien gezogen
ae9e 60                       rts

ae9f 4c 28 a9    nocol4:      jmp nocol

                 ;Indextabelle der 12 Linien fuer BOX
aea2 00 01 02    permuts2:    .by 0,1,2,0,1,5
00 01 05
aea8 00 01 05                 .by 0,1,5,3,4,5
03 04 05
aeae 03 04 05                 .by 3,4,5,3,4,2
03 04 02
aeb4 03 04 02                 .by 3,4,2,0,1,2
00 01 02
aeba 00 01 02                 .by 0,1,2,6,7,8
06 07 08
aec0 06 07 08                 .by 6,7,8,9,10,8
09 0a 08
aec6 09 0a 08                 .by 9,10,8,9,10,11
09 0a 0b
aecc 09 0a 0b                 .by 9,10,11,6,7,11
06 07 0b
aed2 06 07 0b                 .by 6,7,11,6,7,8
06 07 08
aed8 06 07 0b                 .by 6,7,11,0,1,5
00 01 05
aede 03 04 05                 .by 3,4,5,9,10,11
09 0a 0b
aee4 03 04 02                 .by 3,4,2,9,10,8
09 0a 08
```

Listing 4/6.4.4.16 (Teil 2)

4/6.4.4.17

BLOCK X1,Y1,X2,Y2

Beim BLOCK-Befehl bestimmen die beiden Punkte wieder ein Rechteck. Jedoch wird hier nicht einfach dieses Rechteck gezeichnet, sondern es wird der vom Rechteck definierte Bereich gefüllt.

Um dies zu erreichen, werden die Koordinaten zunächst wieder mit <inmem> gespeichert. Ist die Y-Koordinate des zweiten Punktes kleiner als die Y-Koordinate des ersten Punktes, werden die Punkte miteinader vertauscht. Dann wird eine Linie von den so entstandenen Koordinaten (X1,Y1) nach (X2,Y2) gezogen, und Y1 um eins erhöht. Dies macht man solange, bis Y1 größer als Y2 ist. Damit ist der Block gezeichnet.

```
                  ;-----------------------------------------
                  ;---- XBASIC-Befehl: BLOCK X1,Y1,X2,Y2 --
                  ;-----------------------------------------
aeea 20 cd aa     BBblock:   jsr getgkoor     ;Koordinaten 1 holen
aeed a0 00                   ldy #0           ;ab mem speichern
aeef 20 c9 ad                jsr inmem
aef2 20 45 99                jsr chkkom       ;Koordinaten 2 holen
aef5 20 cd aa                jsr getgkoor
aef8 a0 03                   ldy #3           ;ab mem+3 speichern
aefa 20 c9 ad                jsr inmem
aefd ad 3e 03                lda mem+2        ;Y1<Y2
af00 cd 41 03                cmp mem+5
af03 90 13                   bcc onelineb     ;ja
af05 a0 02                   ldy #2           ;nein, Koordinaten tauschen
af07 b9 3c 03     swap:      lda mem,y        ;Wert auf Stack
af0a 48                      pha
af0b b9 3f 03                lda mem+3,y      ;2. Wert speichern
af0e 99 3c 03                sta mem,y
af11 68                      pla              ;1. Wert von Stack
af12 99 3f 03                sta mem+3,y
af15 88                      dey              ;bis 3 Werte vertauscht
af16 10 ef                   bpl swap
af18 a0 05        onelineb:  ldy #5           ;6 Werte setzen
af1a be 35 af     onelinebl: ldx permut,y     ;Index holen
af1d bd 3c 03                lda mem,x        ;Wert holen
af20 99 fa 03                sta xcurs,y      ;Wert speichern
af23 88                      dey              ;bis 6 Werte gesetzt
af24 10 f4                   bpl onelinebl
af26 20 69 ac                jsr line         ;Linie ziehen
af29 ee 3e 03                inc mem+2        ;Y1+1
af2c ad 41 03                lda mem+5        ;Y1>Y2 ?
af2f cd 3e 03                cmp mem+2
af32 b0 e4                   bcs onelineb     ;nein, naechste Linie ziehen
af34 60                      rts

af35 00 01 02     permut:    .by 0,1,2,3,4,2
03 04 02
```

Listing 4/6.4.4.17

4/6.4.4.18

CIRCLE X,Y,RX,RY

Mit dem CIRCLE-Befehl können beliebige Ellipsen oder Kreise gezeichnet werden. Dabei definiert das Paar (X,Y) den Mittelpunkt, und RX bzw. RY den Radius in X- bzw Y-Richtung in Pixeln. Da der Algorithmus vollständig in Integrerarithmetik arbeitet, ist er recht schnell. Außerdem berechnet er nur ein Viertel der Ellipse, und gewinnt die anderen 3/4 durch Spiegelungen. Bei den Berechnungen können aber sehr große Zahlen auftreten. Daher müssen wir uns eine Multiplikationsroutine für 24-Bit Zahlen schreiben. Außerdem sind die Radien auf maximal 165 Pixel begrenzt worden. Dies ergibt aber bereits Ellipsen, von denen nur noch Eckpunkte sichtbar sind. In der Praxis ist dies also keine Einschränkung.

Um den Algorithmus vorzustellen, wird hier das entsprechende BASIC-Programm für CIRCLE 150,100,50,40 angegeben:

```
10  XO=150:YO=100
20  A=50:B=40
30  GOSUB100:END
100 X=O:Y=B
110 QB=2*B*B:QA=2*A*A
120 DX=B*B:DY=QA*B-A*A
130 DA=A*A+B*B
140 PLOT XO+X,YO+Y:PLOT XO-X,YO+Y
150 PLOT XO+X,YO-Y:PLOT XO-X,YO-Y
160 IF DA>=O THEN DA=DA-DX:DX=DX+QB:X=X+1
170 IF DA<O  TEHN DA=DA+DY:DY=DY-QA:Y=Y-1
180 IF DA>O THEN 140
190 PLOT XO-X,YO:PLOT XO+X,YO
```

Listing 4/6.4.4.18 (Teil 1)

```
               ;---------------------------------------
               ;--- XBASIC-Befehl: CIRCLE X,Y,A,B  ---
               ;---------------------------------------
               x0:         .eq mem          ;Mittelpunkt
               y0:         .eq x0+2
               x:          .eq y0+1         ;x Offset
               y:          .eq x+2          ;y Offset
               qa:         .eq y+2          ;Hilfsregister
               qb:         .eq qa+3
               dx:         .eq qb+3
               dy:         .eq dx+3
               da:         .eq dy+3
               a:          .eq da+3         ;Radius A
               b:          .eq a+1          ;Radius B

af3b 4c 28 a9  rbigerr:    jmp nocol        ;Fehler melden

af3e 20 cd aa  BBcircle:   jsr getgkoor     ;Mittelpunkt holen
af41 20 45 99              jsr chkkom
af44 20 55 99              jsr getbyte      ;Radius A holen
af47 e0 a6                 cpx #166         ;max. 165
af49 b0 f0                 bcs rbigerr      ;zu gross
af4b 8e 52 03              stx a            ;Radius A
af4e 20 45 99              jsr chkkom       ;Radius B holen
af51 20 55 99              jsr getbyte
af54 e0 a6                 cpx #166         ;max. 165
af56 b0 e3                 bcs rbigerr      ;zu gross
af58 8e 53 03              stx b            ;Radius B
af5b 8e 41 03              stx y            ;Y=B
af5e ad fa 03              lda xcurs        ;X;X0=X-Koordinate
af61 8d 3c 03              sta x0
af64 ad fb 03              lda xcurs+1
af67 8d 3d 03              sta x0+1
af6a ad fc 03              lda ycurs        ;Y0=Y-Koordinate
af6d 8d 3e 03              sta y0
af70 a9 00                 lda #0           ;High Byte von Y =0
af72 8d 42 03              sta y+1
af75 8d 3f 03              sta x            ;X=0
af78 8d 40 03              sta x+1
af7b 85 63                 sta reg1+2       ;Rechenregister fuer
af7d 85 6b                 sta reg2+2       ;Multiplikation
af7f 85 62                 sta reg1+1       ;=0
af81 85 6a                 sta reg2+1
af83 ad 53 03              lda b            ;B in Register 1 & 2
af86 85 61                 sta reg1
af88 85 69                 sta reg2
af8a 20 58 b1              jsr multi24      ;B*B
af8d 8d 46 03              sta qb           ;QB=B*B
af90 8e 47 03              stx qb+1
af93 8c 48 03              sty qb+2
af96 8d 49 03              sta dx           ;DX=B*B
af99 8e 4a 03              stx dx+1
af9c 8c 4b 03              sty dx+2
af9f 8d 4f 03              sta da           ;DA=B*B
afa2 8e 50 03              stx da+1
afa5 8c 51 03              sty da+2
afa8 0e 46 03              asl qb           ;QB=2*B*B
afab 2e 47 03              rol qb+1
afae 2e 48 03              rol qb+2
afb1 a9 00                 lda #0           ;A in Register 1 & 2
afb3 85 63                 sta reg1+2
afb5 85 6b                 sta reg2+2
afb7 85 62                 sta reg1+1
```

Listing 4/6.4.4.18 (Teil 2)

```
afb9 85 6a     910             sta reg2+1
afbb ad 52 03 911              lda a
afbe 85 61     912             sta reg1
afc0 85 69     913             sta reg2
afc2 20 58 b1 914              jsr multi24         ;A*A
afc5 8d 43 03 915              sta qa              ;QA=A*A
afc8 8e 44 03 916              stx qa+1
afcb 8c 45 03 917              sty qa+2
afce 8d 4c 03 918              sta dy              ;DY=A*A
afd1 8e 4d 03 919              stx dy+1
afd4 8c 4e 03 920              sty dy+2
afd7 0e 43 03 921              asl qa              ;QA=2*A*A
afda 2e 44 03 922              rol qa+1
afdd 2e 45 03 923              rol qa+2
afe0 18        924             clc                 ;DA=A*A+B*B
afe1 6d 4f 03 925              adc da
afe4 8d 4f 03 926              sta da
afe7 8a        927             txa
afe8 6d 50 03 928              adc da+1
afeb 8d 50 03 929              sta da+1
afee 98        930             tya
afef 6d 51 03 931              adc da+2
aff2 8d 51 03 932              sta da+2
aff5 a9 00     933             lda #0              ;B in Register 1
aff7 85 63     934             sta reg1+2
aff9 85 62     935             sta reg1+1
affb ad 53 03 936              lda b
affe 85 61     937             sta reg1
b000 ad 43 03 938              lda qa              ;QA in Register 2
b003 85 69     939             sta reg2
b005 ad 44 03 940              lda qa+1
b008 85 6a     941             sta reg2+1
b00a ad 45 03 942              lda qa+2
b00d 85 6b     943             sta reg2+2
b00f 20 58 b1 944              jsr multi24         ;QA*B
b012 38        945             sec                 ;DY=QA*B-A*A
b013 ed 4c 03 946              sbc dy
b016 8d 4c 03 947              sta dy
b019 8a        948             txa
b01a ed 4d 03 949              sbc dy+1
b01d 8d 4d 03 950              sta dy+1
b020 98        951             tya
b021 ed 4e 03 952              sbc dy+2
b024 8d 4e 03 953              sta dy+2
               954
b027 20 d6 b0 955    circle:   jsr punkt           ;Punkt setzen
b02a 20 42 b1 956              jsr invx            ;X=-X
b02d 20 d6 b0 957              jsr punkt           ;Punkt setzen
b030 20 2c b1 958              jsr invy            ;Y=-Y
b033 20 d6 b0 959              jsr punkt           ;Punkt setzen
b036 20 42 b1 960              jsr invx            ;X=-X
b039 20 d6 b0 961              jsr punkt           ;Punkt setzen
b03c 20 2c b1 962              jsr invy            ;Y=-Y
b03f ad 51 03 963              lda da+2            ;DA<0 ?
b042 30 40     964             bmi negstep         ;ja
b044 38        965             sec                 ;DA=DA-DX
b045 ad 4f 03 966              lda da
b048 ed 49 03 967              sbc dx
b04b 8d 4f 03 968              sta da
b04e ad 50 03 969              lda da+1
b051 ed 4a 03 970              sbc dx+1
b054 8d 50 03 971              sta da+1
```

Listing 4/6.4.4.18 (Teil 3)

```
b057 ad 51 03              lda da+2
b05a ed 4b 03              sbc dx+2
b05d 8d 51 03              sta da+2
b060 18                    clc                  ;DX=DX+QB
b061 ad 49 03              lda dx
b064 6d 46 03              adc qb
b067 8d 49 03              sta dx
b06a ad 4a 03              lda dx+1
b06d 6d 47 03              adc qb+1
b070 8d 4a 03              sta dx+1
b073 ad 4b 03              lda dx+2
b076 6d 48 03              adc qb+2
b079 8d 4b 03              sta dx+2
b07c ee 3f 03              inc x                ;X=X+1
b07f d0 03                 bne negstep
b081 ee 40 03              inc x+1
b084 ad 51 03  negstep:    lda da+2             ;DA>=0 ?
b087 10 3b                 bpl nonegstep        ;ja
b089 18                    clc
b08a ad 4f 03              lda da               ;DA=DA+DY
b08d 6d 4c 03              adc dy
b090 8d 4f 03              sta da
b093 ad 50 03              lda da+1
b096 6d 4d 03              adc dy+1
b099 8d 50 03              sta da+1
b09c ad 51 03              lda da+2
b09f 6d 4e 03              adc dy+2
b0a2 8d 51 03              sta da+2
b0a5 38                    sec
b0a6 ad 4c 03              lda dy               ;DY=DY-QA
b0a9 ed 43 03              sbc qa
b0ac 8d 4c 03              sta dy
b0af ad 4d 03              lda dy+1
b0b2 ed 44 03              sbc qa+1
b0b5 8d 4d 03              sta dy+1
b0b8 ad 4e 03              lda dy+2
b0bb ed 45 03              sbc qa+2
b0be 8d 4e 03              sta dy+2
b0c1 ce 41 03              dec y                ;Y=Y-1
b0c4 ad 41 03  nonegstep:  lda y                ;Y=0 ?
b0c7 f0 03                 beq endcir           ;ja
b0c9 4c 27 b0              jmp circle           ;nein
b0cc 20 d6 b0  endcir:     jsr punkt            ;Punkt setzen
b0cf 20 42 b1              jsr invx             ;X=-X
b0d2 20 d6 b0              jsr punkt            ;Punkt setzen
b0d5 60                    rts

b0d6 18        punkt:      clc                  ;Punkt setzen
b0d7 ad 3c 03              lda x0               ;X-Koordinate = X0+X
b0da 6d 3f 03              adc x
b0dd 8d fa 03              sta xcurs
b0e0 ad 3d 03              lda x0+1
b0e3 6d 40 03              adc x+1
b0e6 8d fb 03              sta xcurs+1
b0e9 18                    clc                  ;Y-Koordinate = Y0+Y
b0ea ad 3e 03              lda y0
b0ed 6d 41 03              adc y
b0f0 8d fc 03              sta ycurs
b0f3 a9 00                 lda #0
b0f5 6d 42 03              adc y+1
b0f8 d0 47                 bne noplot
b0fa 20 fe b0              jsr clipplot         ;Punkt setzen
b0fd 60                    rts
```

Listing 4/6.4.4.18 (Teil 4)

```
b0fe ad fc 03  clipplot:  lda ycurs        ;Punkt setzen,falls sichtbar
b101 c9 c8                cmp #200
b103 b0 3c                bcs noplot       ;Y>199
b105 2c f8 03             bit colmode
b108 30 13                bmi clipcol      ;Farbmodus
b10a ad fb 03             lda xcurs+1      ;X>319 ?
b10d f0 0b                beq dorealy      ;nein
b10f c9 02                cmp #2
b111 b0 2e                bcs noplot       ;ja
b113 ad fa 03             lda xcurs
b116 c9 40                cmp #<320
b118 b0 27                bcs noplot       ;ja
b11a 4c 81 ab  dorealy:   jmp plot         ;Punkt setzen

b11d ad fb 03  clipcol:   lda xcurs+1      ;X>159 ?
b120 d0 1f                bne noplot       ;ja
b122 ad fa 03             lda xcurs
b125 c9 a0                cmp #160
b127 b0 18                bcs noplot       ;ja
b129 4c 81 ab             jmp plot         ;Punkt setzen

b12c 18        invy:      clc              ;Y=-Y
b12d ad 41 03             lda y            ;Low invertieren
b130 49 ff                eor #$ff
b132 69 01                adc #1           ;+1
b134 8d 41 03             sta y
b137 ad 42 03             lda y+1
b13a 49 ff                eor #$ff         ;High invertieren
b13c 69 00                adc #0           ;+Carry
b13e 8d 42 03             sta y+1
b141 60        noplot:    rts

b142 18        invx:      clc              ;X=-X
b143 ad 3f 03             lda x            ;Low invertieren
b146 49 ff                eor #$ff
b148 69 01                adc #1           ;+1
b14a 8d 3f 03             sta x
b14d ad 40 03             lda x+1
b150 49 ff                eor #$ff         ;High invertieren
b152 69 00                adc #0           ;+Carry
b154 8d 40 03             sta x+1
b157 60                   rts

               ;-----------------------------------------
               ;--- 24-Bit Multiplikation             ---
               ;--- reg1*reg2->merg
               ;-----------------------------------------
b158 a9 00     multi24:   lda #0           ;Ergebnis =0
b15a 85 6c                sta merg
b15c 85 6d                sta merg+1
b15e 85 6e                sta merg+2
b160 a2 18                ldx #24          ;24 Bit Zaehler
b162 06 6c     multipl:   asl merg         ;Ergebnis *2
b164 26 6d                rol merg+1
b166 26 6e                rol merg+2
b168 06 61                asl reg1         ;Register1 *2
b16a 26 62                rol reg1+1
b16c 26 63                rol reg1+2
b16e 90 13                bcc unaddr2      ;hoechstes Bit war nicht gesetzt
b170 18                   clc              ;Ergebnis=Ergebnis+Register2
b171 a5 6c                lda merg
b173 65 69                adc reg2
b175 85 6c                sta merg
```

Listing 4/6.4.4.18 (Teil 5)

```
b177 a5 6d              lda merg+1
b179 65 6a              adc reg2+1
b17b 85 6d              sta merg+1
b17d a5 6e              lda merg+2
b17f 65 6b              adc reg2+2
b181 85 6e              sta merg+2
b183 ca      unaddr2:   dex             ;Zaehler-1
b184 d0 dc              bne multipl     ;bis 24 Bit abgearbeitet
b186 a5 6c              lda merg        ;Ergebnis in Y/X/A
b188 a6 6d              ldx merg+1
b18a a4 6e              ldy merg+2
b18c 60                 rts
```

Listing 4/6.4.4.18 (Teil 6)

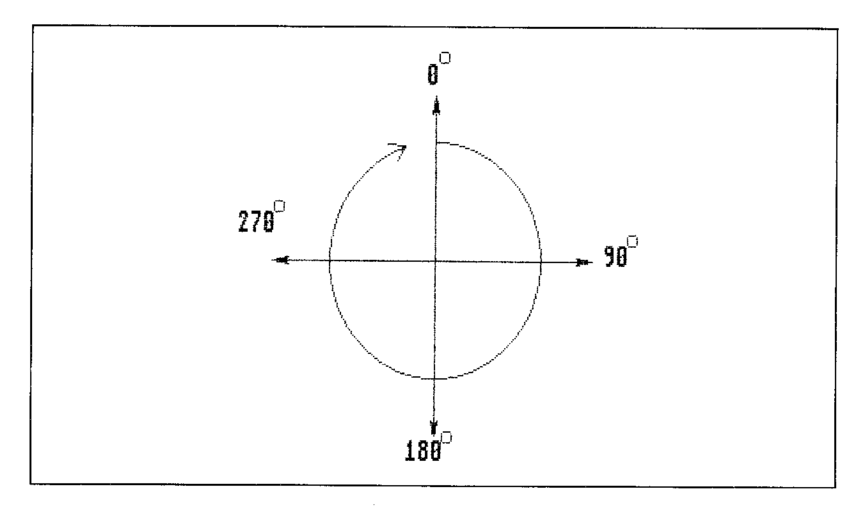
4/6.4.4.19

ANGLE X,Y,RX,RY,AW,EW,SW

Der ANGLE-Befehl ist wohl der vielseitigste Befehl dieser Grafikerweiterung, aber auch leider der langsamste Befehl. Mit ihm ist es möglich, Ellipsensegmente unter Angabe der Schrittweite zu zeichnen. Damit ist es möglich, Kreise, Ellipsen, Ellipsensegmente, aber auch Dreiecke, Vierecke, Fünfecke usw. zu zeichnen.

Man sollte darauf achten, das die Linien innerhalb des Bildschirms bleiben, da sonst die Punkte auf den sichtbaren Bereich projiziert werden, und zwar nur mit der Koordinate, die nicht mehr auf den Bildschirm paßt. Die führt zu Verzerrungen, die sich mit steigender Schrittweite immer stärker auswirken.

Die Parameter X, Y, RX und RY beschreiben die Ellipse, wie wir es von dem CIRCLE-Befehl her bereits kennen. Der Parameter AW gibt den Startwinkel an, und der Parameter EW den Entwinkel. Mit dem Parameter SW gibt man die Schrittweite an, um die der Winkel erhöht wird. Dabei gilt das folgende Winkelsystem:



Der Pfeil gibt die Zeichenrichtung an.

Mit dem folgenden BASIC-Programm soll der Algorithmus wieder verdeutlicht werden: (für ANGLE 160,100,150,90,359,180,10)

```
10 X=160:Y=100
20 RX=150:RY=90
30 AW=359:EW=180:SW=10
40 GOSUB 100
50 END
60
100 GOSUB200:PLOT XW,YW
101 IF AW>EW THEN EW=EW+360
102 GRAFON 7,0
110 AW=AW+SW:IF AW>EW THEN AW=EW
120 GOSUB200:LINE XW,YW
130 IF AW=EW THEN END
140 GOTO110
150
200 XW=X+SIN(2*3.14159/360*AW)*RX
210 YW=Y-COS(2*3.14159/360*AW)*RY
220 RETURN
```

Listing 4/6.4.4.19 (Teil 1)

```
                    ;------------------------------------------------
                    ;---- Befehl: ANGLE X,Y,RX,RY,AW,EW,SW  -
                    ;------------------------------------------------
                    ;x0,y0 wie bei circle

                    lineflag:   .eq y0+1            ;Flag fuer Linie ziehen
                    rx:         .eq lineflag+1      ;Radius X
                    ry:         .eq rx+1            ;Radius Y
                    aw:         .eq ry+1            ;Anfangswinkel
                    ew:         .eq aw+2            ;Endwinkel
                    sw:         .eq ew+2            ;Schrittweite (Winkel)

b18d 20 cd aa       BBangle:    jsr getgkoor        ;Mittelpunkt holen
b190 ad fa 03                   lda xcurs           ;in X0 und
b193 8d 3c 03                   sta x0
b196 ad fb 03                   lda xcurs+1
b199 8d 3d 03                   sta x0+1
b19c ad fc 03                   lda ycurs           ;Y0 speichern
b19f 8d 3e 03                   sta y0
b1a2 20 45 99                   jsr chkkom          ;Radius X holen
b1a5 20 55 99                   jsr getbyte
b1a8 8e 40 03                   stx rx              ;und speichern
b1ab 20 45 99                   jsr chkkom
b1ae 20 55 99                   jsr getbyte         ;Radius Y holen
b1b1 8e 41 03                   stx ry              ;und speichern
b1b4 20 45 99                   jsr chkkom
b1b7 20 5d 99                   jsr getadr          ;Anfangswinkel holen
b1ba a5 14                      lda $14             ;und speichern
b1bc a6 15                      ldx $15
b1be 8d 42 03                   sta aw
b1c1 8e 43 03                   stx aw+1
b1c4 20 45 99                   jsr chkkom          ;Endwinkel holen
b1c7 20 5d 99                   jsr getadr
b1ca a5 14                      lda $14             ;und speichern
b1cc a6 15                      ldx $15
b1ce 8d 44 03                   sta ew
b1d1 8e 45 03                   stx ew+1
b1d4 ec 43 03                   cpx aw+1            ;Anfangswinkel>=Endwinkel ?
b1d7 f0 15                      beq vgl2w
b1d9 b0 1a                      bcs nextarg         ;nein
b1db 18             add360:     clc                 ;Endwinkel +360 (Grad)
b1dc ad 44 03                   lda ew
b1df 69 68                      adc #<360
b1e1 8d 44 03                   sta ew
b1e4 ad 45 03                   lda ew+1
b1e7 69 01                      adc #>360
b1e9 8d 45 03                   sta ew+1
b1ec 90 07                      bcc nextarg         ;Schritt lesen
b1ee cd 42 03       vgl2w:      cmp aw              ;Anfangswinkel>=Endwinkel ?
b1f1 f0 e8                      beq add360          ;ja
b1f3 90 e6                      bcc add360          ;ja
b1f5 20 45 99       nextarg:    jsr chkkom          ;Schrittweite lesen
b1f8 20 55 99                   jsr getbyte
b1fb 8e 46 03                   stx sw              ;und speichern
b1fe a9 00                      lda #0              ;Lineflag auf Null
b200 8d 3f 03                   sta lineflag

b203 20 5b b2       sp:         jsr makepoint       ;Punkt berechnen
b206 2c 3f 03                   bit lineflag        ;Lineflag gesetzt ?
b209 30 08                      bmi lineit          ;ja, Linie ziehen
b20b 20 81 ab                   jsr plot            ;nein, ersten Punkt plotten
b20e ce 3f 03                   dec lineflag        ;Flag setzen
b211 30 03                      bmi linecont        ;unbedingter Sprung
b213 20 69 ac       lineit:     jsr line            ;Linie ziehen
```

Listing 4/6.4.4.19 (Teil 2)

```
b216 ad 42 03  linecont:  lda aw         ;Anfangswinkel=Endwinkel ?
b219 cd 44 03             cmp ew
b21c d0 08                bne continag   ;nein
b21e ad 43 03             lda aw+1
b221 cd 45 03             cmp ew+1
b224 f0 34                beq fertigag   ;ja, fertig
b226 18        continag:  clc            ;Anfangswinkel+Schrittweite
b227 ad 42 03             lda aw
b22a 6d 46 03             adc sw
b22d 8d 42 03             sta aw
b230 ad 43 03             lda aw+1
b233 69 00                adc #0
b235 8d 43 03             sta aw+1
b238 cd 45 03             cmp ew+1       ;AW>=EW ?
b23b f0 11                beq vglb2
b23d 90 c4                bcc sp         ;nein, naechsten Punkt
b23f ad 44 03  makelast:  lda ew         ;AW=EW (letzter Punkt)
b242 8d 42 03             sta aw
b245 ad 45 03             lda ew+1
b248 8d 43 03             sta aw+1
b24b 4c 03 b2  tosp:      jmp sp         ;Punkt setzen
b24e ad 42 03  vglb2:     lda aw         ;AW<=EW ?
b251 cd 44 03             cmp ew
b254 f0 f5                beq tosp       ;ja AW=EW
b256 90 f3                bcc tosp       ;ja AW<EW
b258 b0 e5                bcs makelast   ; AW>EW
b25a 60        fertigag:  rts

b25b ad fa 03  makepoint: lda xcurs      ;alten Grafikcursor retten
b25e 8d fd 03             sta xcurs2
b261 ad fb 03             lda xcurs+1
b264 8d fe 03             sta xcurs2+1
b267 ad fc 03             lda ycurs
b26a 8d ff 03             sta ycurs2
b26d ac 42 03             ldy aw
b270 ad 43 03             lda aw+1
b273 20 53 a1             jsr ergebnis   ;FAC=AW
b276 a9 7b                lda #<pifac
b278 a0 9d                ldy #>pifac
b27a a2 32                ldx #50        ;FAC=AW*2*PI/360
b27c 20 d8 98             jsr basicrom
b27f a2 38                ldx #56
b281 20 d8 98             jsr basicrom   ;FAC=SIN(FAC)
b284 a2 34                ldx #52
b286 20 d8 98             jsr basicrom   ;FAC->ARG
b289 a9 00                lda #0
b28b ac 40 03             ldy rx
b28e 20 53 a1             jsr ergebnis   ;FAC=RX
b291 a2 36                ldx #54
b293 20 d8 98             jsr basicrom   ;FAC=RX*SIN(AW*2*PI/360)
b296 a2 3c                ldx #60
b298 20 d8 98             jsr basicrom   ;in Integer wandeln
b29b 18                   clc            ;X0 + Integerwert
b29c aa                   tax            ;ist X-Koordinate
b29d 98                   tya
b29e 6d 3c 03             adc x0
b2a1 8d fa 03             sta xcurs
b2a4 8a                   txa
b2a5 6d 3d 03             adc x0+1
b2a8 8d fb 03             sta xcurs+1
b2ab 30 73                bmi clipxm     ;<0, X auf 0 setzen
b2ad 2c f8 03             bit colmode    ;Farbmodus ?
b2b0 30 11                bmi colclipx   ;ja
b2b2 c9 00                cmp #0         ;X<320 ?
```

Listing 4/6.4.4.19 (Teil 3)

```
b2b4 f0 18     1240            beq xunclip       ;ja
b2b6 c9 02     1241            cmp #2
b2b8 b0 6f     1242            bcs clipxp1       ;nein, X auf 319 setzen
b2ba ad fa 03  1243            lda xcurs
b2bd c9 40     1244            cmp #<320
b2bf b0 68     1245            bcs clipxp1       ;nein, X auf 319 setzen
b2c1 90 0b     1246            bcc xunclip       ;unbedingter Sprung
b2c3 c9 00     1247  colclipx: cmp #0            ;X<160 ?
b2c5 d0 6d     1248            bne clipxp2       ;nein, X auf 159 setzen
b2c7 ad fa 03  1249            lda xcurs
b2ca c9 a0     1250            cmp #160
b2cc b0 66     1251            bcs clipxp2       ;nein, X auf 159 setzen
               1252
b2ce ac 42 03  1253  xunclip:  ldy aw
b2d1 ad 43 03  1254            lda aw+1
b2d4 20 53 a1  1255            jsr ergebnis      ;FAC=AW
b2d7 a9 7b     1256            lda #<pifac
b2d9 a0 9d     1257            ldy #>pifac
b2db a2 32     1258            ldx #50           ;FAC=AW*2*PI/360
b2dd 20 d8 98  1259            jsr basicrom
b2e0 a2 3a     1260            ldx #58
b2e2 20 d8 98  1261            jsr basicrom      ;FAC=COS(FAC)
b2e5 a2 34     1262            ldx #52
b2e7 20 d8 98  1263            jsr basicrom      ;FAC->ARG
b2ea a9 00     1264            lda #0
b2ec ac 41 03  1265            ldy ry
b2ef 20 53 a1  1266            jsr ergebnis      ;FAC=RY
b2f2 a2 36     1267            ldx #54
b2f4 20 d8 98  1268            jsr basicrom      ;FAC=RY*COS(AW*2*PI/360)
b2f7 a2 3c     1269            ldx #60
b2f9 20 d8 98  1270            jsr basicrom      ;nach Integer wandeln
b2fc 38        1271            sec               ;Y Koordinate =
b2fd 84 14     1272            sty $14           ;Y0-Integerwert
b2ff 85 15     1273            sta $15
b301 ad 3e 03  1274            lda y0
b304 e5 14     1275            sbc $14
b306 8d fc 03  1276            sta ycurs
b309 a9 00     1277            lda #0
b30b e5 15     1278            sbc $15
b30d 30 0b     1279            bmi clipym        ;Y<0 -> Y auf Null setzen
b30f ad fc 03  1280            lda ycurs
b312 c9 c8     1281            cmp #200
b314 b0 01     1282            bcs clipyp        ;Y>199 -> Y auf 199 setzen
b316 60        1283            rts
               1284
b317 a9 c7     1285  clipyp:   lda #199          ;Y=199
b319 2c        1286            bit
b31a a9 00     1287  clipym:   lda #0            ;Y=0
b31c 8d fc 03  1288            sta ycurs
b31f 60        1289            rts
b320 a9 00     1290  clipxm:   lda #0            ;X=0
b322 8d fa 03  1291            sta xcurs
b325 8d fb 03  1292            sta xcurs+1
b328 60        1293            rts
b329 a9 3f     1294  clipxp1:  lda #<319         ;X=319
b32b 8d fa 03  1295            sta xcurs
b32e a9 01     1296            lda #>319
b330 8d fb 03  1297            sta xcurs+1
b333 60        1298            rts
b334 a9 9f     1299  clipxp2:  lda #<159         ;X=159
b336 8d fa 03  1300            sta xcurs
b339 a9 00     1301            lda #>159
b33b 8d fb 03  1302            sta xcurs+1
b33e 60        1303            rts
               1304
```

Listing 4/6.4.4.19 (Teil 4)

4/6.4.4.20

COLBLOCK X1,Y1,X2,Y2,ZF,HF

Mit dem Befehl COLBLOCK ist es möglich, für Teilbereiche des Bildschirms die Zeichen- und Hintergrundfarbe neu festzulegen. Da in der hochauflösenden Grafik für jeweils 8*8 Pixel ein Byte in „farbehires" die Zeichen- und Hintergrundfarbe angibt, kann man alle 16 Farben des C 64 gleichzeitig auf dem Bildschirm darstellen. Mit COLBLOCK wird ein Bereich definiert, für den die neue Zeichenfarbe ZF, und die neue Hintergrundfarbe HF gilt.

Die Koordinaten werden dabei als Textkoordinaten angegeben, da immer nur 8*8 Pixel große Felder neu bestimmt werden können. Dies entspricht genau den Textkoordinaten. Außerdem muß die erste Koordinate die linke obere Ecke, und die zweite Koordinate die rechte untere Ecke bestimmen. Die Koordinaten müssen also folgende Bedingungen erfüllen:

```
X1 < 40 & X2 < 40
X1 < X2 & Y1 < Y2
Y1 < 25 & Y2 < 25
```

Die Routine holt zuerst die vier Parameter und speichert sie ab „mem". Dabei wird überprüft, ob die oben aufgestellten Bedingungen erfüllt sind. Ist dies nicht der Fall, wird der Fehler „Illegal Quantity" gemeldet.

Sonst kann der Block gesetzt werden. Dazu wird zunächst der Zeilenanfang der oberen Zeile nach der Formel „azg=farbehires+40*Y1" berechnet, wobei die Multiplikation mit 40 durch eine Tabelle ersetzt wird. Das Y-Register durchläuft dann die Werte von X2 bis X1, so daß die obere Zeile mit „STA (azg),Y" gesetzt werden kann. Dann wird der Zeiger „azg" um 40 erhöht, also eine Zeile tiefer gesetzt. Das Verfahren wird solange fortgesetzt, bis die letzte Zeile gesetzt wurde.

```
                    ;-----------------------------------------
                    ;--Befehl: COLBLOCK X1,Y1,X2,Y2,ZF,HF --
                    ;-----------------------------------------

b33f 4c 28 a9       nocol5:    jmp nocol          ;Fehler melden

b342 20 55 99       BBcolblock:jsr getbyte        ;Koordinate X1 holen
b345 8e 3c 03                  stx mem            ;speichern
b348 e0 28                     cpx #40            ;<40 ?
b34a b0 f3                     bcs nocol5         ;nein, Fehler
b34c 20 45 99                  jsr chkkom
b34f 20 55 99                  jsr getbyte        ;Koordinate Y1 holen
b352 8e 3d 03                  stx mem+1          ;speichern
b355 e0 19                     cpx #25            ;<25 ?
b357 b0 e6                     bcs nocol5         ;nein, Fehler
b359 20 45 99                  jsr chkkom
b35c 20 55 99                  jsr getbyte        ;Koordinate X2 holen
b35f 8e 3e 03                  stx mem+2          ;speichern
b362 ec 3c 03                  cpx mem            ;<X1 ?
b365 90 d8                     bcc nocol5         ;ja, Fehler
b367 e0 28                     cpx #40            ;<40 ?
b369 b0 d4                     bcs nocol5         ;nein, Fehler
b36b 20 45 99                  jsr chkkom
b36e 20 55 99                  jsr getbyte        ;Koordinate Y2 holen
b371 8e 3f 03                  stx mem+3          ;speichern
b374 ec 3d 03                  cpx mem+1          ;<Y1 ?
b377 90 c6                     bcc nocol5         ;ja, Fehler
b379 e0 19                     cpx #25            ;<25 ?
b37b b0 c2                     bcs nocol5         ;nein, Fehler
b37d 20 45 99                  jsr chkkom
b380 20 55 99                  jsr getbyte        ;Zeichenfarbe holen
b383 e0 10                     cpx #16            ;gueltig ?
b385 b0 b8                     bcs nocol5         ;nein, zu gross
b387 86 fc                     stx azg+1          ;merken
b389 20 45 99                  jsr chkkom         ;auf Komma testen
b38c 20 55 99                  jsr getbyte        ;Hintergrundfarbe holen
b38f e0 10                     cpx #16            ;gueltig ?
b391 8a                        txa
b392 b0 ab                     bcs nocol5         ;nein, zu gross
b394 06 fc                     asl azg+1          ;Zeichenfarbe * 16
b396 06 fc                     asl azg+1
b398 06 fc                     asl azg+1
b39a 06 fc                     asl azg+1
b39c 05 fc                     ora azg+1          ;+Hintergrundfarbe
b39e 48                        pha                ;Farbwert merken
b39f ad 3d 03                  lda mem+1
b3a2 0a                        asl                ;Y0*2
b3a3 a8                        tay                ;in Y
b3a4 18                        clc
b3a5 a9 00                     lda #<farbehires   ;azg auf Farbram setzen
b3a7 79 e0 b3                  adc mult40,y       ;+40*Y0
b3aa 85 fb                     sta azg
b3ac a9 c8                     lda #>farbehires
b3ae 79 e1 b3                  adc mult40+1,y
b3b1 85 fc                     sta azg+1
b3b3 68                        pla                ;Farbwert zurueckholen
b3b4 ac 3e 03       setcolbl:  ldy mem+2          ;Y=Y1
b3b7 91 fb          setcolb:   sta (azg),y        ;Farbwert setzen
b3b9 88                        dey                ;Y-1
b3ba 30 05                     bmi rd1            ;setzen, bis Y<Y0
b3bc cc 3c 03                  cpy mem
b3bf b0 f6                     bcs setcolb
b3c1 ce 3f 03       rd1:       dec mem+3          ;X1-1
```

Listing 4/6.4.4.20 (Teil 1)

```
b3c4 ae 3f 03 1368              ldx mem+3          ;fertig, wenn X1<X0
b3c7 30 16    1369              bmi readycb
b3c9 ec 3d 03 1370              cpx mem+1
b3cc 90 11    1371              bcc readycb
b3ce 48       1372              pha                ;Farbwert merken
b3cf 18       1373              clc
b3d0 a5 fb    1374              lda azg            ;azg+40 (in naechste Zeile)
b3d2 69 28    1375              adc #40
b3d4 85 fb    1376              sta azg
b3d6 a5 fc    1377              lda azg+1
b3d8 69 00    1378              adc #0
b3da 85 fc    1379              sta azg+1
b3dc 68       1380              pla
b3dd 90 d5    1381              bcc setcolbl       ;unbedingter Sprung
b3df 60       1382  readycb:    rts
              1383
              1384  ;Multiplikationstabelle x*40, x von 0 bis 24
b3e0 00 00 28 1385  mult40:     .wo 0,40,80,40*3,40*4,40*5,40*6,40*7,40*8,40*9,40*10
00 50 00 78 00 a0 00 c8 00 f0 00 18 01 40 01 68 01 90 01
b3f6 b8 01 e0 1386              .wo 40*11,40*12,40*13,40*14,40*15,40*16,40*17,40*18
01 08 02 30 02 58 02 80 02 a8 02 d0 02
b406 f8 02 20 1387              .wo 40*19,40*20,40*21,40*22,40*23,40*24
03 48 03 70 03 98 03 c0 03
              1388
              1389
```

Listing 4/6.4.4.20 (Teil 2)

4/6.4.9

Übersicht der neuen Basic-Befehle

Um Ihnen das Nachschlagen zu erleichtern, drucken wir nachfolgend eine alphabetische Übersicht der neuen Befehle (Stand: XBasic 1.1) ab.

Befehl Funktion	Bedeutung
AT(Z,S)	setze den Cursor auf Zeile Z, Spalte S (liefert " " als Funktionsergebnis)
AUTO STEP	automatische Zeilennumerierung
C 64	schaltet die Basic-Erweiterung neu aus
CLS	löscht den Bildschirm
COLLECT	räumt die Diskette auf ('V'-Befehl)
DEC(X$)	liefert den Dezimalwert der Hexzahl X$
DEEK (AD)	wie PEEK, jedoch 16-Bit breit (Doppel-PEEK)
DEFCHAR X, Z, M, A$	Definiert das Zeichen X aus dem Zeichensatz Z neu. A$ enthält die Definition und M den Aufbau der Definition
DIR MASKE$	Ausgabe des Disketteninhalts

Befehl Funktion	Bedeutung
DOKE, AD,W	wie POKE, jedoch 16 Bit breit (Doppel-POKE)
DREEK (AD)	Wie DEEK, arbeitet jedoch immer im RAM (Doppel-RAM-PEEK)
DROKE, AD, W	wie DOKE, arbeitet aber immer im RAM (Doppel-RAM-POKE)
FRE (X)	wie die Originalfunktion
HEADER FORM$	formatieren der Diskette
HEX$ (X)	liefert X als vierstellige Hexzahl
INITIALISE	initialisieren der Diskette ('I'-Befehl)
INKEY$	wartet auf einen Tastendruck und liefert die Taste als String. Der Cursor wird dabei eingeschaltet.
INST E$, I$, P	überschreibt die Zeichenkette I$ ab Position P mit der Zeichenkette E$
INSTR(P,S$,IN$)	sucht den String S$ in IN$ ab Position P
LINEINPUT A$	eine Zeichenkette einlesen
LINEINPUT#D, A$	eine Zeichenkette aus einer Datei lesen
OLD	stellt ein Programm nach NEW wieder her

Befehl Funktion	Bedeutung
OLDCHAR	kopiert den Originalzeichensatz in das RAM
PAUSE TIME	PAUSE-Funktion
QLOAD FILES$, DEF	laden einer ASCII-Datei als Programm
REEK (AD)	wie PEEK, arbeitet jedoch immer im RAM (RAM-PEEK)
RENAME A$ TO N$	umbenennen von Dateien auf Diskette
ROKE AD, W	wie POKE, arbeitet jedoch immer im RAM (RAM-POKE)
SCRATCH FILES$	löschen von Dateien auf der Diskette
SETDATNR	setzt interne Dateinummer für Disk-Befehle
SETEOL X	Ende-Kennung für LINEINPUT# festlegen
SETTIME UHR, ZT$	Uhrzeit setzen
SPACE$ (X)	liefert X Leerzeichen
ST$	liest den Fehlerkanal der Floppy
STRINGS$ (A$,X)	liefert x-mal den String A$
SWAP A, B	vertauschen der Variableninhalte

Befehl Funktion	Bedeutung
TIME X	Uhrzeit lesen
XBASIC	startet die Basic-Erweiterung neu

4/6.5

Basic-Erweiterungen beim C 128

(Autor: Werner Eberl)

Der Commodore 128 besitzt zwar viele neue praktische Befehle, jedoch kann man hier die in Computerkreisen übliche Erfahrung bestätigen, daß kein Programm so gut sein kann, daß man es nicht verbessern könnte. Das trifft also auch das Basic 7.0 zu. Ergänzungen bieten sich vor allem bei der Zahlkonvertierung an (die Verwendung von hexadezimalen Operanden im Basictext ist noch etwas umständlich), bei der Verwendung der Echtzeit-Uhren und bei der Ein-/Ausgabe.

Außerdem hat wohl jeder Anwender sein Spezialgebiet, und gerade für diese Gebiete bräuchte man dann spezielle Befehle, um sich das Leben mit dem Basic-Programmieren zu erleichtern. Deshalb stellen wir auch im ersten Teil dieses Kapitels ein Verfahren vor, wie man allgemein neue Befehle und Funktionen in das Basic 7.0 eingliedert. Es wurde darauf geachtet, daß der Speicherbereich für die Basic-Erweiterungen so gelegt wurde, daß auch noch ROM-Routinen verwendet werden können.

In der vorliegenden Version liegen die neuen Basic-Befehle und Funktionen in der Bank 1 zwischen $0400 und $1000. Die obere Grenze kann jedoch leicht auf $4000 verschoben werden, so daß dann 15 KByte für Ihre Basic-Erweiterungen zur Verfügung stehen.

4/6.5.1

Allgemeine Vorgehensweise

Nach einer Liste der wichtigsten Adressen des Basic-Interpreters und des Betriebssystems gehen wir ausführlich auf das Bank-Switching bei den Basic-Erweiterungen ein. Danach wird beschrieben, wie man die Basic-Erweiterung aktiviert, und im nächsten Kapitel folgt die Erläuterung der Dekodierung von Befehlen und Funk-

tionen. Die in diesem Buch realisierten neuen Befehle finden Sie dann im folgenden Kapitel zusammengestellt. Dort ist auch beschrieben, bei welchen Speicherbereichen welche Befehlsgruppen abgelegt worden sind. Den Abschluß bildet ein Kapitel über die Möglichkeiten, Parameter an Basic-Befehle oder Funktionen zu übergeben. Gerade das zweite und das letzte Unterkapitel beinhalten viele Informationen, die Ihnen langes Herumprobieren und Fast-Verzweifeln ersparen.

4/6.5.1.1

Symbolvereinbarungen

Im folgenden zeigen wir Ihnen eine Liste mit den wichtigsten Basic-Routinen und Adressen.

```
                .BA $1300
       ;          .OU "P128.OBJ"
       ; BASIC-ADRESSEN
       VALTYP:  .EQ $0F          ;TYPFLAG STRING/NUM.
       POKER :  .EQ $16          ;FUER FACADR
       INDEX :  .EQ $24          ;ZEIGER AUF STRING
       TXTPTR:  .EQ $3D          ;BASIC-BEFEHLS-ZEIGER
       VARPNT:  .EQ $47          ;VARIABLENADRESSE
       FACEXP:  .EQ $63          ;EXPONENT DES FAC
       FACSGN:  .EQ $68          ;VORZEICHEN DES FAC
       ;
       ; ADRESSEN DER BASIC-VEKTOREN
       IERROR:  .EQ $0300        ;FEHLERMELDUNG
       IMAIN :  .EQ $0302        ;BASIC-WARMSTART
       ICRNCH:  .EQ $0304        ;TOKENISATION
       IQPLOP:  .EQ $0306        ;LIST-ROUTINE
       IGONE :  .EQ $0308        ;BEFEHL AUSFUEREN
       IEVAL :  .EQ $030A        ;ARITHM. ELEM. AUSW.
       IESCLK:  .EQ $030C        ;ESC-TOKENISATION
       IESCPR:  .EQ $030E        ;ESC-LIST
       IESCEX:  .EQ $0310        ;ESC-AUSFUEHREN
       USRPOK:  .EQ $1218        ;ADR. D. USERVEKTORS
       ;
       ; STANDARD-BASIC-ROUTINEN
       NGONE:   .EQ $4AA2        ;BASIC-BEFEHL BEARB.
       NEVAL :  .EQ $78DA        ;ARITHM. ELEM. AUSW.
       NIRQ  :  .EQ $FA65        ;STANDARD-IRQ-ROUTINE
       ;
```

Listing 4/6.5.1.1-1 (Teil 1)

```
; BASIC-ZEIGER
TXTTAB:     .EQ $002D       ;BEGINN BASIC-TEXT
VARTAB:     .EQ $002F       ;ANFANG VARIABLE
ARYTAB:     .EQ $0031       ;BEGINN DER ARRAYS
STREND:     .EQ $0033       ;ENDE ARRAYS + 1
FRETOP:     .EQ $0035       ;ANFANG DER STRINGS
MAXMEM1:    .EQ $0039       ;ENDE BASIC-RAM BANK1
;
; PARAMETER-HOL-ROUTINEN
CHRGET:     .EQ $0380       ;EIN BASIC-Z. LESEN
CHRGOT:     .EQ $0386       ;LETZEN KODE LESEN
GETWRD:     .EQ $8812       ;ADRESSPARAM. HOLEN
CHKOPN:     .EQ $7959       ;KLAMMER AUF PRUEFEN
CHKCLS:     .EQ $7956       ;KLAMMER ZU PRUEFEN
FRMNUM:     .EQ $77D7       ;HOLT NUM. AUSDRUCK
CHKNUM:     .EQ $77DA       ;PRUEFT AUF NUMERISCH
PARCHK:     .EQ $7950       ;HOLT AUSDRUCK IN ()
FRESTR:     .EQ $877E       ;HOLT STRINGATTR.
FRMSTR:     .EQ $877B       ;HOLT STRING
FRMEVL:     .EQ $77EF       ;HOLT AUSDRUCK
CHKCOM:     .EQ $795C       ;PRUFT AUF KOMMA
GETBYT:     .EQ $87F4       ;HOLT BYTE-PARAMETER
GTBYTC:     .EQ $87F1       ;CHRGET & GETBYT
COMBYT:     .EQ $8809       ;CHKCOM & GETBYT
PLSV:       .EQ $91AE       ;LOAD/SAVE-PARAMETER
GETNUM:     .EQ $8803       ;FRMNUM,GETADR,COMBYT
PTRGET:     .EQ $B08B       ;SUCHT VARIABLE
;
; ARITHMETISCHE ROUTINEN
SNGFLT:     .EQ $84D4       ;WANDELT Y -> FAC
GIVAYF:     .EQ $793C       ;WANDELT A/Y -> FAC
FLOATS:     .EQ $8C70       ;WAND. -> FAC; X=EXP.
GETADR:     .EQ $8815       ;FAC NACH POKER (L,H)
FMULT  :    .EQ $8A24       ;FAC MAL KONSTANTE
MOVMF  :    .EQ $8C00       ;FAC NACH VARIABLE
NOSFLT:     .EQ $84C9       ;(A/Y) -> FAC (>0)
FLPINT:     .EQ $849F       ;FAC -> (A/Y) (+/-)
QINT   :    .EQ $8CC7       ;QUICK INT-FUNCTION
;
; BASIC-STRINGVERWALTUNG
GETSPA:     .EQ $9299       ;PLATZ F. NEUEN STR.
PUTNEW:     .EQ $86E3       ;DES. AUF STRINGSTACK
STRLIT:     .EQ $869A       ;STRING AUS PUFFER H.
;
; BASIC-ROUTINEN FUER EIN-AUSGABE
STROUT:     .EQ $55E2       ;GIBT STRING AUS
COOUT  :    .EQ $90E8       ;SETZT EINGABEDATEI
COIN   :    .EQ $90FD       ;SETZT AUSGABEDATEI
CGETL  :    .EQ $9109       ;HOLT EIN ZEICHEN
OUTCH  :    .EQ $90DF       ;GIBT EIN ZEICHEN AUS
INCHR  :    .EQ $90E5       ;ZEICHEN EINLESEN
LINPRT:     .EQ $8E32       ;ADRESSE AUSGEBEN
REALSP:     .EQ $5604       ;LEERZEICHEN AUSGEBEN
;
; SONSTIGE BASIC-ROUTINEN
NEWSTT:     .EQ $4AF6       ;INTERPRETERSCHLEIFE
ERROR  :    .EQ $4D3C       ;FEHLERMELDUNG AUSG.
```

Listing 4/6.5.1.1-1 (Teil 2)

```
SNERR   :    .EQ $796C        ;'SYNTAX ERROR'
FCERR   :    .EQ $7D28        ;'ILL. QUANTITY ERR.'
;
; BANK-SWITCH-ROUTINEN/ADRESSEN
INDTXT:      .EQ $03C9        ;HOLE ZEICH. AUS BASIC
INDIN1R1:    .EQ $03B7        ;RAM1 VON RAM1 LESEN
CURRENTB:    .EQ $03D5        ;AKTUELLE BANKNR.
;
SWRAM0  :    .EQ $FF01
SWRAM1  :    .EQ $FF02
SWROMRA0:    .EQ $FF03
SWROMRA1:    .EQ $FF04
;
RUECKH:      .EQ $03          ;ADRESSEN ZUM UEBERLISTEN DES
RUECKL:      .EQ $B2          ;RTS-BEFEHLS AM ENDE EINER
;                               ROM-ROUTINE BEI AUFRUF
;                               VON BANK 1 AUS
;
```

Listing 4/6.5.1.1-1 (Teil 3)

Ebenso wichtig sind die Betriebssystem-Adressen des Kernal. Die Routinen mit Adressen $FF. . werden über die konstante Kernalsprungtabelle angesprochen, die aufwärtskompatibel zum Commodore 64 ist und wahrscheinlich auch bei künftigen Commodore-Rechnern erhalten bleiben wird.

```
ACPTR   :    .EQ $FFA5        ; BYTE VOM SER. BUS
CHKIN   :    .EQ $FFC6        ; KANAL F. EINGABE
CHKOUT  :    .EQ $FFC9        ; KANAL F. AUSGABE
CHRIN   :    .EQ $FFCF        ; BYTE VOM KANAL HOLEN
CHROUT  :    .EQ $FFD2        ; BYTE AUF KANAL AUSG.
CIOUT   :    .EQ $FFA8        ; BYTE AUSGEBEN
CLOSE   :    .EQ $FFC3        ; DATEI SCHLIESSEN
CLRCHN  :    .EQ $FFCC        ; KANAL ABMELDEN
GETIN   :    .EQ $FFE4        ; ZEICHEN HOLEN
LISTEN  :    .EQ $FFB1        ; LISTEN-KOMMANDO
LOAD    :    .EQ $FFD5        ; LADE-ROUTINE
OPEN    :    .EQ $FFC0        ; DATEI OEFFNEN
READST  :    .EQ $FFB7        ; STATUS LESEN
SAVE    :    .EQ $FFD8        ; SPEICHER-ROUTINE
SECOND  :    .EQ $FF93        ; SEK.ADR. NACH LISTEN
SETLFS  :    .EQ $FFBA        ; SETZTE FILENAMENPAR.
STOP    :    .EQ $FFE1        ; PRUEFT STOPTASTE
TALK    :    .EQ $FFB4        ; TALK SENDEN
TKSA    :    .EQ $FF96        ; SEK.ADR NACH TALK
UNLSN   :    .EQ $FFAE        ; UNLISTEN SENDEN
```

Listing 4/6.5.1.1-2 (Teil 1)

```
UNTALK:        .EQ $FFAB          ; UNTALK SENDEN
SETBNK:        .EQ $FF68          ; BANK F. LOAD/SAVE/V
GETCFG:        .EQ $FFCB          ; BANK-NR. -> MMU-WERT
INDFET:        .EQ $FF74          ; LESEN V. AND. BANK
INDSTA:        .EQ $FF77          ; SCHR. IN AND. BANK
INDCMP:        .EQ $FF7A          ; VGL. MIT AND. BANK
; KERNAL-VEKTOREN
IIRQ   :       .EQ  $0314         ;IRQ-VEKTOR
;
; SPEZIELLE KERNAL-VARIABLE
FETVEC:        .EQ $02AA          ; F. ADR. F. INDFET
STAVEC:        .EQ $02B9          ; F. ADR. F. INDSTA
;
; SYSTEMADRESSEN
MMUCR  :       .EQ $FF00          ;MMU CONFIG.REG.
;
```

Listing 4/6.5.1.1-2 (Teil 2)

Da wir in diesem Kapitel nur den Sprungverteiler auf die einzelnen Befehle besprechen wollen, hängt das Verständnis des Verteilers nicht davon ab, wo die einzelnen Befehle nun genau liegen. Trotzdem muß der Verteiler selbst natürlich wissen, wohin er springen muß. Deshalb folgt hier die Liste der Adressen der Sprungziele.

```
HEXCONV:       .EQ $0400
BINCONV:       .EQ $0403
BINSTR:        .EQ $0406
VALNEU  :      .EQ $0409
;
AUS:           .EQ $0800
DOKE:          .EQ $0803
PAUSE:         .EQ $0806
DEEK:          .EQ $0809
SETTIME:       .EQ $080C
TIME:          .EQ $080F
TEST:          .EQ $0812
;
LEST:          .EQ $0B00
LESP:          .EQ $0B03
VIEW:          .EQ $0B06
;
```

Listing 4/6.5.1.1-3

4/6.5.1.2

Bank-Switching bei Basic-Erweiterungen

Worum es beim Bank-Switching überhaupt geht, wurde bereits bei der Besprechung der MMU erwähnt. Um die Problematik für die Basic-Erweiterungen transparenter zu machen, bilden wir hier zunächst eine Grafik ab, aus der Sie ersehen können, in welcher Bank und in welchem Adreßbereich die für die neuen Befehle relevanten Bereiche liegen. Die Banknummern bezeichnen dabei nicht die Konfigurationsnummern, die mit dem BANK-Befehl eingestellt werden können, sondern die physikalischen Bänke. Die fünfzehn Möglichkeiten des BANK-Befehles grafisch dargestellt finden Sie in Teil 3.

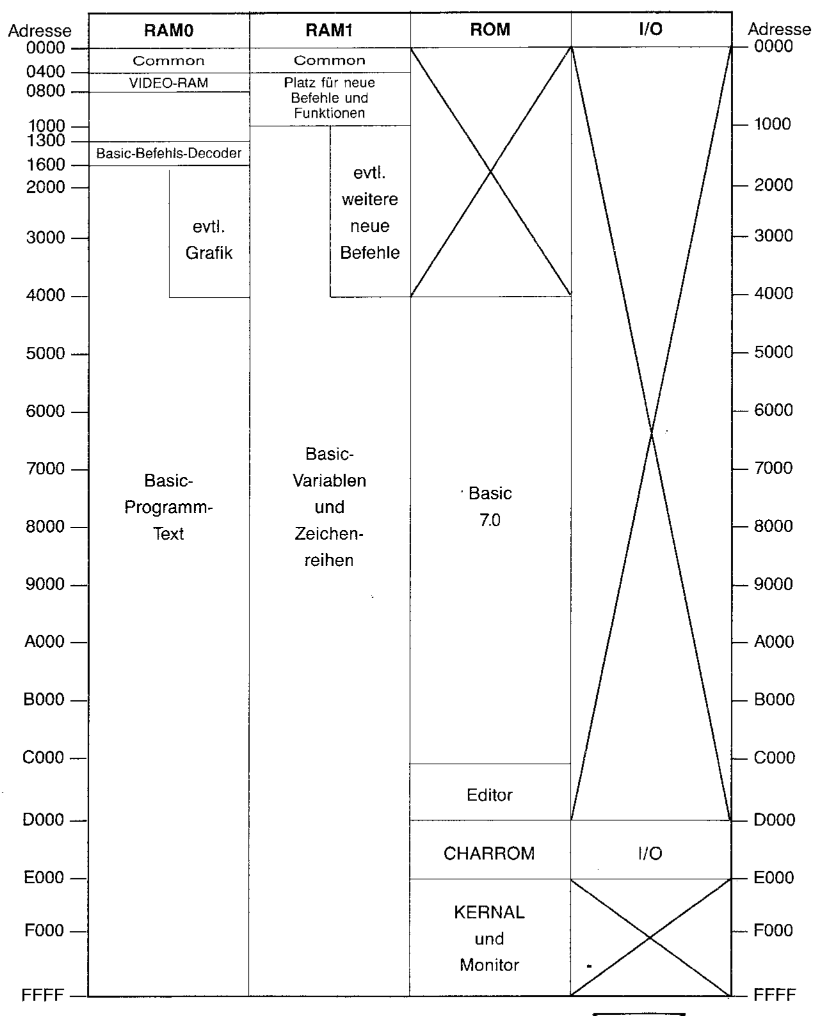


Bild 4/6.5.1.2-1 Speicheraufteilung beim C 128 mit den Erweiterungen ⊠ : nicht existent

Die Präkonfigurations-Register der MMU werden von Basic-Interpreter bei dessen Initialisierung folgendermaßen besetzt:

Name	Adresse	Konfiguration
RAM0	$FF01	nur RAM 0, kein ROM, kein I/O
RAM1	$FF02	nur RAM 1, kein ROM, kein I/O
ROMRAM0	$FF03	RAM 0, ab $4000 ROM, kein I/O
ROMRAM1	$FF04	RAM 1, ab $4000 ROM, kein I/O

Jetzt wollen wir die für uns wichtigsten Punkte zusammentragen:

1. Unsere Basic-Erweiterungen sollen ladbar und veränderbar sein und müssen deshalb im RAM stehen.

2. Wir wollen einige Basic-ROM-Routinen und vor allem auch die Kernal-Routinen von unseren Basic-Erweiterungen aus ansprechen können, ohne dabei mittlere Klimmzüge machen zu müssen.

3. Aus den Punkten 1 und 2 folgt, daß die oberen 48 KB des Prozessor-Adreßraums für unsere Zwecke ausscheiden, weil sich dort ROM befindet.

4. Der verbleibende Platz in der RAM-Bank 0 geht v on $1300 bis zum Beginn des Basic-Programms und ist daher recht knapp. Die eigentlichen Programme für die neuen Basic-Befehle müssen wir daher in RAM-Bank 1 legen.

5. Die ersten 1 KB der RAM-Bank 1 scheiden aus, da dieser Adreßbereich durch die MMU zur Common-Area erklärt wird, d.h. hier ist immer RAM 0 aktiv.

6. Den notwendigen Platz für die Befehle in RAM-Bank 1 reservieren wir durch Veränderung des Zeigers auf den Anfang der Variablentabelle. Maximal können wir dadurch 15 KByte für unsere Zwecke nutzbar machen.

7. Die Dekodierung unserer eigenen Befehle erreichen wir, indem wir den Vektor auf die Befehlsausführung bzw. auf die Auswertung eines arithmetischen Ausdrucks entsprechend verbiegen. Die Vektoren selbst stehen in der Common-Area, beinhalten aber keine Information, in welcher Bank die Basic-Erweiterungen stehen.

8. Wenn der Basic-Interpreter einen indirekten Sprung über einen dieser Vektoren ausführt, so ist die im Basic-Interpreter am meisten verwendete Konfiguration aktiv, nämlich:

RAM 0	ROM	
$000	$4000	$FFFF

Bild 4/6.5.1.2-2 Standard-Basic-Konfiguration „ROMRAM0"

Abgekürzt bezeichnet man diese Konfiguration mit ROMRAM0. Die Tatsache, daß bei der Verzweigung auf unsere Routine eben diese Konfiguration aktiv ist, zwingt uns, wenigstens einen Teil unserer Basic-Erweiterungen, am besten den Sprungverteiler, in die RAM-Bank 0 zu legen, auch wenn dort der Platz etwas knapp ist.

9. Bemerkenswert ist, daß keine der durch die Präkonfigurations-Register einstellbaren Möglichkeiten einen Zugriff auf den I/O-Bereich zuläßt.
10. Solange eine Konfiguration mit ROM oder I/O eingeschaltet ist, kann weder auf den gesamten Basic-Programmtext, noch auf die gesamten Basic-Variablen bzw. Strings zugegriffen werden.
11. Für Basic-Erweiterungen sind die ROM-Routinen zum Parameter holen sehr wichtig. Diese Parameterhol-Routinen greifen jedoch auf den Basictext zu und müssen deshalb Umkonfigurationen vornehmen. Es ist damit zu rechnen, daß nach Abschluß dieser Routine die Basic-Standardkonfiguration ROMRAM0 aktiv ist, auch wenn wir vorher ROMRAM1 konfiguriert haben. Dies gilt besonders für die Routine CHRGET, die in der Common-Area abgelegt ist.
12. Beim Auswerten von arithmetischen Ausdrücken können wieder arithmetische Ausdrücke auftreten, wodurch sich verschachtelte Aufrufe unserer eigentlichen Arithmetik-Routine ergeben können. Vor dem zweiten Aufruf der Arithmetik-Routine kann das Basic auf ROMRAM0 zurückkonfiguriert haben oder nicht. Weil dieser Punkt nicht vorhersehbar ist, müssen wir den Sprung auf unsere Arithmetik-Routine in die Common-Area legen.

Aus dieser Tabelle der zu beachtenden Punkte folgt, daß wir uns beim Programmieren der Basic-Erweiterungen auf munteres Umkonfigurieren einstellen müssen. Gott sei Dank können wir die geläufigsten und wichtigsten Konfigurationen ja durch Verwendung der Präkonfigurations-Register benützen, so daß uns dann wenigstens keine Prozessor-Register verloren gehen. Bei Verwendung des Konfigurations-Registers bei $FF00 müßten wir ja wenigstens eines der Register benützen, um aus diesem die neue Konfiguration in dieses MMU-Register zu schreiben.

Weitsprünge, die nicht im Sand verlaufen

Die hier vorgestellte Routine führt einen sogenannten Weitsprung aus. Damit ist gemeint, daß außer dem Programmzähler noch die Banknummer verändert wird. In der vorliegenden Routine wurde der Spezialfall verwirklicht, daß das Sprungziel in der Konfiguration ROMRAM1 enthalten ist und das aufrufende Programm in der Konfiguration ROMRAM0. Das Byte $20 ist der JSR-Befehl, auf den die 2 Bytes mit der Zieladresse folgen. Die Zieladresse wird vor dem Aufruf der Weitsprung-Routine vom aufrufenden Programm hier eingesetzt. Das aufgerufene Unterprogramm kehrt sicher in die Weitsprung-Routine zurück, da diese in der Common-Area steht. Zum Abschluß der Routine wird auf ROMRAM0 zurückkonfiguriert, und damit ist das Unterprogramm auch fertig.

```
                 202   WEITCODE:
1349 8D 04 FF    203                STA SWROMRA1      ;KOFIG. ROM/RAM1
134C 20          204                .BY $20           ;JSR-BEFEHL
134D 1A 13       205   WCZ:         .WO IDUMMY        ;ZIELADRESSE
134F 8D 00 FF    206                STA SWROMRA0    , .RUECKKONFIGURIEREN
1352 60          207                RTS
                 208   ;
```

Listing 4/6.5.1.2-1

Hilfssprung auf die Arithmetik-Routine

```
                 209   HEVALCODE:
1353 8D 03 FF    210                STA SWROMRA0
1356 4C FF 13    211                JMP EEVAL
                 212   ;
```

Listing 4/6.5.1.2-2

Wie erwähnt, muß der Sprung in die Arithmetik-Routine in der Common-Area liegen, weil eventuell umkonfiguriert werden muß. Dies ist hier deutlich zu sehen. Durch den STA-Befehl wird auf ROMRAM0 konfiguriert und dann zu unserer eigentlichen Arithmetik-Routine verzweigt.

Umrahmte Basic-Parameter-Routinen

Das Problem der Parameter-Routinen ist bereits erwähnt worden. Wir führen uns nochmal die Problematik vor Augen: Es möge eine Basic-Erweiterung in der Konfiguration ROMRAM1 laufen. Diese ruft eine Parameterhol-Routine auf, die ihrerseits auf ROMRAM0 umkonfiguriert. Wenn diese Routine den RTS-Befehl ausführt, springt sie ins Leere, da die aufrufende Routine ja in RAM 1 gestanden hat. Im Prinzip könnten wir nun hergehen und für jede Basic-Routine eine umrahmte Routine in der Common-Area ablegen. Für die GETBYT-Routine würde das dann wie folgt aussehen:

```
ZGETBYT:
JSR GETBYT
STA SWROMRA1
RTS
```

Die Routine würde sicher zurückkehren, da ja der Aufruf in der Common-Area liegt; danach würde auf ROMRAM1 zurückkonfiguriert und alles wäre in Ordnung.

Wir müßten dann jeweils anstatt GETBYT die erweiterte Routine ZGETBYT aufrufen.

Der uns zur Verfügung stehende Platz in der Common-Area ist aber äußerst gering. Er beschränkt sich auf 6 Byte in der Zero-Page und etwa 23 Byte in Page 3. In dem Bereich der Page 3 haben wir bereits die Weitsprung-Routine und den Hilfssprung

für die Arithmetik-Routine gelegt. Deshalb wählen wir hier einen anderen Weg. Das Zurückkonfigurieren auf ROMRAM1 und der RTS-Befehl befinden sich schon vom Betriebssystem her initialisiert in der Common-Area und zwar ab der Adresse $03B3. Wir müssen jetzt nur erreichen, daß die ROM-Routine kurz dorthin springt, bevor sie zu unserem aufrufenden Programm zurückkehrt. Das machen wir mit folgendem kleinen Trick, der hier wieder am Beispiel der Routine GETBYT erläutert sei:

```
                190   EGETBYT:
0439 A9 03      191              LDA #RUECKH
043B 48         192              PHA
043C A9 B2      193              LDA #RUECKL
043E 48         194              PHA
043F 4C F4 87   195              JMP GETBYT
                196   ;
```

Listing 4/6.5.1.2-3

Diese Umrahmung der ROM-Routine steht im RAM 1. Dadurch können wir sehr viele ROM-Routinen auf diese Weise umrahmen, ohne daß der Platz ausgeht.

Der Trick ist nun folgender: Wir legen als Rückkehradresse die Adresse des kleinen Programmstücks ab $03B3 auf den Stapel. Danach wird die ROM-Routine aufgerufen. Wenn diese auf den RTS-Befehl läuft, meint sie, sie wäre von dem Programm bei $03B3 aufgerufen worden und springt deshalb dahin zurück. Dort wird dann umkonfiguriert und alles ist in Butter.

Wenn man ganz genau sein will, muß man natürlich anmerken, daß die um eins verminderte Rücksprungadresse auf den Stapel gelegt werden muß, da der RTS-Befehl den Programmzähler selbständig um 1 erhöht.

Wir können nun die ROM-Routine aufrufen, indem wir anstatt der Adresse im ROM die Adresse unserer umrahmten Routine in der RAM-Bank 1 verwenden, also z.B. EGETBYT anstatt GETBYT. Diejenigen ROM-Routinen, die nicht auf den Basic-Programmspeicher zugreifen, können ohne diese Umrahmung aufgerufen werden. Das sind vor allem die Routinen für die Fließkomma-Arithmetik und alle Kernal-Routinen. Es ist auch unkritisch, die Fehlerbehandlungsroutinen anzuspringen, weil dort die Banks in geeigneter Weise geschaltet werden.

Der Übersichtlichkeit halber wollen wir Ihnen den gesamten Programmteil mit den umrahmten ROM-Routinen auflisten:

```
                ;************************************
                ;* ROM-ROUTINEN MIT SWITCH          *
                ;* ------------------------------   *
                ;* BEI AUFRUF EINER ROM-ROUTINE,    *
                ;* DIE AUF DEN BASICTEXT ZUGREIFT*
                ;* IST STATT DER NORMALEN ADRESSE*
                ;* DIE ADRESSE DER FOLGENDEN ER-    *
                ;* WEITERUNG ZU WAEHLEN !           *
                ;************************************
                ;
                ECHKCOM:
041E A9 03              LDA #RUECKH
0420 48                 PHA
0421 A9 B2              LDA #RUECKL
0423 48                 PHA
0424 4C 5C 79           JMP CHKCOM
                ;
                EPARCHK:
0427 A9 03              LDA #RUECKH
0429 48                 PHA
042A A9 B2              LDA #RUECKL
042C 48                 PHA
042D 4C 50 79           JMP PARCHK
                ;
                EFRMEVL:
0430 A9 03              LDA #RUECKH
0432 48                 PHA
0433 A9 B2              LDA #RUECKL
0435 48                 PHA
0436 4C EF 77           JMP FRMEVL
                ;
                EGETBYT:
0439 A9 03              LDA #RUECKH
043B 48                 PHA
043C A9 B2              LDA #RUECKL
043E 48                 PHA
043F 4C F4 87           JMP GETBYT
                ;
                EGETWRD:
0442 A9 03              LDA #RUECKH
0444 48                 PHA
0445 A9 B2              LDA #RUECKL
0447 48                 PHA
0448 4C 12 88           JMP GETWRD
                ;
                EGETNUM:
044B A9 03              LDA #RUECKH
044D 48                 PHA
044E A9 B2              LDA #RUECKL
0450 48                 PHA
0451 4C 03 88           JMP GETNUM
                ;
                ECHRGET:
0454 A9 03              LDA #RUECKH
0456 48                 PHA
0457 A9 B2              LDA #RUECKL
0459 48                 PHA
045A 4C 54 04           JMP ECHRGET
                ;
```

Listing 4/6.5.1.2-4

4/6.5.1.3

Initialisierung

```
1300 4C B8 14                 JMP INIT
                   WEITCODE:
1349 8D 04 FF                 STA SWROMRA1       ;KOFIG. ROM/RAM1
134C 20                       .BY $20            ;JSR-BEFEHL
134D 1A 13         WCZ:       .WO IDUMMY         ;ZIELADRESSE
134F 8D 03 FF                 STA SWROMRA0       ;RUECKKONFIGURIEREN
1352 60                       RTS
                   ;
                   HEVALCODE:
1353 8D 03 FF                 STA SWROMRA0
1356 4C FF 13                 JMP EEVAL
                   ;
                   WCL:       .EQ *-WEITCODE     ;PROGRAMMLAENGE
                   WEITSPR:   .EQ $0400-WCL      ;IN DIE ERSTEN 1K
                   HEVAL:     .EQ $0400-6
                   WCO:       .EQ WCZ-WEITCODE   ;OFFSET
                   ZIEL:      .EQ WEITSPR+WCO    ;EFF.ADR V ZIEL
                   ;
                   WCODECOP:
1359 A2 10                    LDX #WCL           ;ANZAHL BYTES
                   WCC1:
135B BD 48 13                 LDA WEITCODE-1,X   ;WEITCODE BYTE
135E 9D EF 03                 STA WEITSPR-1,X    ;FUER BYTE KOPIE-
1361 CA                       DEX                ;REN; SCHLEIFE
1362 D0 F7                    BNE WCC1
1364 60                       RTS                ;FERTIG
                   ;
                   ;INITIALISIERUNG
                   ;
                   INIT:
14B8 20 59 13                 JSR WCODECOP       ;WEITCODE KOPIEREN
                   ;
14BB A9 70                    LDA #<EGONE        ;IGONE-VECTOR AUF
14BD 8D 08 03                 STA IGONE          ;VERTEILER FUER NEUE
14C0 A9 13                    LDA #>EGONE        ;BEFEHLE STELLEN
14C2 8D 09 03                 STA IGONE+1
                   ;
14C5 A9 FA                    LDA #<HEVAL        ;ARTITHM.-VEKTOR AUF
14C7 8D 0A 03                 STA IEVAL          ;AUSWERTUNG VON $/%-
14CA A9 03                    LDA #>HEVAL        ;AUSDRUECKEN STELLEN
14CC 8D 0B 03                 STA IEVAL+1
                   ;
14CF A9 10                    LDA #$10           ;BEGINN DER VARIAB-
14D1 85 30                    STA VARTAB+1       ;LENTAB. AUF $11000
                   ;
14D3 78                       SEI
14D4 A9 1B                    LDA #<EIRQ         ;IRQ-VEKTOR AUF
14D6 8D 14 03                 STA IIRQ           ;MANAGER STELLEN
14D9 A9 13                    LDA #>EIRQ
```

Listing 4/6.5.1.3-1 (Teil 1)

```
14DB 8D 15 03 439           STA IIRQ+1
14DE 58       440           CLI
              441    ;
14DF A9 E7    442           LDA #<MELDUNG     ;ZEIGER (A/Y) AUF
14E1 A0 14    443           LDY #>MELDUNG     ;MELDUNG
14E3 20 E2 55 444           JSR STROUT        ;STRING AUSGEBEN
14E6 60       445           RTS               ;INITIALISIERUNG FERTIG
              446    ;
14E7 45 52 57 447    MELDUNG: .BY "ERW. C128"
2E 20 43 31 32 38
14F0 0D       448           .BY $0D
14F1 00       449           .BY 0
              450    ;
```

Listing 4/6.5.1.3-1 (Teil 2)

Die Initialisierungs-Routine liegt in der RAM-Bank 0 ab $1300=4864. Sie wird deshalb mit

```
BANK0: SYS4864
```

aufgerufen. Den Bank-Befehl kann man sich sparen, wenn der Rechner seit dem Einschalten keinen anderen Bank-Befehl erhalten hat.

Zu Beginn des Programms steht ein Sprung auf den wirklichen Beginn der Initialisierungs-Routine, wodurch Flexibilität bei Programmänderungen gewährleistet ist. Die abgebildeten Hilfszellen BEFNR, FNNR und AKKUSICH werden für die Befehls- und Funktionsdekodierung benötigt und werden hier gleich mit abgebildet.

Das Programmstück ab dem Label WEITCODE wird von der Initialisierungs-Routine, genauer gesagt, von dem Unterprogramm WCODECOP so in die Common-Area kopiert, daß das letzte Byte gerade bei $03FF zu liegen kommt. Die Symbolzuweisungen darunter legen die wirklichen Adressen der Unterprogrammlabels in der Common-Area fest. Dies sind:

WEITSPR	für die Weitsprung-Routine
HEVAL	für den Sprung auf die Arithmetik-Routine
ZIEL	für die Adresse des Weitsprungvektors

Die eigentliche Init-Routine beginnt beim Label INIT. Dort wird zunächst die Weitsprung-Routine in die Common-Area kopiert, dann werden die drei Vektoren zum Ausführen eines Befehls (IGONE), zum Auswerten eines arithmetischen Ausdrucks (IEVAL) und zur Ausführung des 60 Hertz-Interrupts (IIRQ) auf unsere eigenen Routinen gestellt.

Zwischendurch wird noch der Zeiger auf den Beginn der Variablen-Tabelle auf $1000 gestellt, so daß die Basic-Varibalen nicht unsere Basic-Erweiterungen zerstören.

Zum Schluß wird noch eine kleine Meldung ausgegeben, wobei die Basic-Routine STROUT verwendet wird, die einen String, dessen Anfang in Akku und Y-Register übergeben wurde, solange ausgibt, bis sie auf ein 0-Byte trifft.

4/6.5.1.4

Dekodierung

Zur Behandlung der normalen Basic-Befehlswörter in einer Befehlszeile führt der Interpreter vier verschiedene Arbeitsschritte durch:

1. Bei der Eingabe der Zeile werden die Schlüsselwörter in sogenannte Tokens umgewandelt, die eine Abkürzung des Basic-Befehlswortes darstellen.

2. Beim LIST-Befehl werden diese Abkürzungen zur Ausgabe auf dem Bildschirm wieder in den ganzen Text umgewandelt.

3. Die Basic-Befehle werden an ihren Tokens erkannt, und danach wird über eine Sprungtabelle zu den einzelnen Befehls-Routinen verzweigt.

4. Ebenso wird mit den Funktionen verfahren, wobei hier allerdings nicht beim Abarbeiten von Befehlen sondern beim Auswerten von arithmetischen Ausdrücken ein Sprungverteiler angesprochen wird.

Im Prinzip könnten wir bei der Realisierung eigener Basic-Befehle genauso verfahren, aber wir können uns die ersten beiden Schritte auch schenken, wenn wir bei der Ausführung von Befehlen und Auswerten von arithmetischen Ausdrücken nachsehen, ob eines unserer Schlüsselwörter im Basic-Text steht.

Dieses Befehlswort ist dann nicht in Tokens umgewandelt und wir vergleichen einfach den Klartext mit unserer Schlüsselworttabelle.

Auf diese Weise funktionieren auch die im folgenden beschriebenen Befehls- und Funktionsdekoder. Sie können die Befehlstabelle leicht ergänzen, wenn Sie einen Assembler haben. Sie müssen dann natürlich auch die Sprungtabelle um ihre Einsprungadresse erweitern. Für diejenigen unter Ihnen, die keinen Assembler verfügbar haben, haben wir einen Testbefehl und eine Testfunktion eingefügt, die man mit Hilfe des Monitors leicht auf die speziellen Bedürfnisse abändern kann.

4/6.5.1.4.1

Befehle

```
                 DECBZ:                             ;BEFEHLSZEIGER -1
1365 A5 3D                  LDA TXTPTR
1367 D0 02                  BNE DECBZ1              ;LOW BYTE <> 0
1369 C6 3E                  DEC TXTPTR+1            ;SONST HIGH-BYTE DEC.
                 DECBZ1:
136B C6 3D                  DEC TXTPTR              ;LOW-BYTE DECREMEN-
136D 60                     RTS                     ;TIEREN UND FERTIG
                 ;
```

Listing 4/6.5.1.4-1

Zuerst möchten wir eine kleine Hilfsroutine abbilden, die einfach den Basic-Befehlszeiger TXTPTR um eins vermindert. Die Routine ist notwendig, weil unsere Dekodierungs-Routine ein Zeichen mit CHRGET liest und dabei den Befehlszeiger um eins weiterstellt. Wurde aber nun doch kein neuer Basic-Befehl gefunden, so muß diese Änderung rückgängig gemacht werden.

```
136E 00          AKKUSICH:  .BY 0                   ;ZWISCHENSPEICHERZELLE
136F 00          BEFNR:     .BY 0                   ;NUMMER DES BEFEHLS
                 EGONE:                             ; ** NEUE BEFEHLE **
                 ;              ; ** AUSFUEHREN      **
1370 20 80 03               JSR CHRGET              ;BASIC-ZEICHEN HOLEN
1373 90 21                  BCC BEBAS               ;CODE WAR ZIFFER
1375 C9 60                  CMP #$60                ;TEST AUF BUCHSTABE
1377 B0 1D                  BCS BEBAS               ;BASIC-BEFEHLS-CODE
1379 C9 41                  CMP #$41                ;VERGLEICH MIT "A"
137B 90 18                  BCC BEBASC              ;SONDERZEICHEN
137D 8D 6E 13               STA AKKUSICH            ;AKKU SICHERN
1380 A2 00                  LDX #0
1382 8E 6F 13               STX BEFNR               ;BEFNR =0
1385 8D 01 FF               STA SWRAM0              ;FUER BASIC-TEXT
                 EG1:
1388 A0 00                  LDY #0
138A EE 6F 13               INC BEFNR               ;BEFNR=BEFNR+1
138D BD E1 13               LDA BTAB,X              ;ZEICH. AUS BEF.TAB.
1390 D0 0D                  BNE EG2                 ;KEIN TRENNZEICHEN
1392 AD 6E 13               LDA AKKUSICH            ;AKKU RESTAURIEREN
                 BEBASC:
1395 38                     SEC                     ;CARRY WIEDER SETZEN
                 BEBAS:
1396 8D 03 FF               STA SWROMRA0            ;STANDARDKONFIG.
1399 20 65 13               JSR DECBZ               ;BEFEHLSZEIGER RUECK
139C 4C A2 4A               JMP NGONE               ;ZUR STANDARDROUTINE
                 EG2:
139F D1 3D                  CMP (TXTPTR),Y          ;VERGL. M. BASIC-TEXT
```

Listing 4/6.5.1.2-2 (Teil 1)

```
13A1 D0 2A                    BNE EG4          ;KEINE UEBEREINST.
13A3 C8                       INY              ;NEXT ZEI. BASIC-TEXT
13A4 E8                       INX              ;NEXT ZEI. BEF.TAB.
13A5 BD E1 13                 LDA BTAB,X       ;ZEICHEN AUS BEF.TAB.
13A8 D0 F5                    BNE EG2          ;NEXT ZEICH. PRUEFEN
13AA 18                       CLC
13AB 98                       TYA
13AC 65 3D                    ADC TXTPTR       ;BEFEHLSZEIGER UM
13AE 85 3D                    STA TXTPTR       ;BEF.-LAENGE ERHOEH.
13B0 90 02                    BCC EG3          ;KEIN UEBERTRAG
13B2 E6 3E                    INC TXTPTR+1     ;UEBERTRAG
                   EG3:
13B4 AD 6F 13                 LDA BEFNR        ;BEFEHLSNUMMER HOLEN
13B7 0A                       ASL              ;VERDOPPELN UND ALS
13B8 AA                       TAX              ;ZEIGER IN SPRUNGTAB.
13B9 CA                       DEX
13BA CA                       DEX
13BB BD D7 13                 LDA STAB,X
13BE 8D F4 03                 STA ZIEL
13C1 BD D8 13                 LDA STAB+1,X
13C4 8D F5 03                 STA ZIEL+1       ;SPRUNGZIEL
13C7 20 F0 03                 JSR WEITSPR
13CA 4C F6 4A                 JMP NEWSTT       ;ZURUECK ZUR
                   ;             INTERPRETERSCHLEIFE
                   ;
                   EG4:
13CD E8                       INX              ;NEXT ZEI. BEF.TAB.
13CE BD E1 13                 LDA BTAB,X
13D1 D0 FA                    BNE EG4          ;BEFEHL NICHT ZU ENDE
13D3 E8                       INX              ;'0' UEBERLESEN
13D4 4C 88 13                 JMP EG1          ;NEXT BEFEHL CHECKEN
                   ;
                   ;SPRUNGTABELLE BEFEHLE
                   ;
                   STAB:
13D7 00 08                    .WO AUS          ;AUS
13D9 03 08                    .WO QOKE         ;QOKE
13DB 06 08                    .WO PAUSE        ;PAUSE
13DD 0C 08                    .WO SETTIME      ;SETTIME
13DF 12 08                    .WO TEST         ;TEST-BEFEHL
                   ;
                   BTAB:
13E1 41 55 53                 .BY "AUS",0
00
13E5 51 4F 4B                 .BY "QOKE",0
45 00
13EA 50 41 55                 .BY "PAUSE",0
53 45 00
13F0 53 45 54                 .BY "SETTIME",0
54 49 4D 45 00
13F8 54 45 53                 .BY "TEST",0
54 00
13FD 00                       .BY 0            ;ENDE BEFEHLSTABELLE
                   ;
```

Listing 4/6.5.1.4-2 (Teil 2)

Nach dem Holen des ersten Basic-Zeichens wird zur ROM-Routine verzweigt, wenn eine Ziffer oder ein Sonderzeichen gefunden wurde. Nur bei einem Buchstaben wird die Befehls-Tabelle Zeichen für Zeichen durchsucht, wobei die Befehlswörter durch ein einzelnes 0-Byte getrennt sind, und die Tabelle durch zwei aufeinanderfolgende 0-Bytes beendet ist. Wurde eines unserer Befehlsworte gefunden, so wird der Basic-Befehlszeiger um die Länge des Befehlswortes erhöht, die Zieladresse aus der Sprungzieltabelle geholt und in den Vektor ZIEL geschrieben, womit die Weitsprung-Routine aufgerufen wird. Die Befehlssequenz endet mit dem Sprung in die Interpreterschleife NEWSTT.

Korrespondierend zu der Sprungtabelle in Bank 0 existiert die Sprungtabelle in Bank 1:

```
                453     ; *** SPRUNGLEISTE ***
0800 4C 15 08   454             JMP AUS
0803 4C 3A 08   455             JMP OOKE
0806 4C 8D 08   456             JMP PAUSE
0809 4C 61 08   457             JMP QEEK
080C 4C DB 08   458             JMP SETTIME
080F 4C 81 09   459             JMP TIME
0812 4C FD 09   460             JMP TEST
                461     ;
```

Listing 4/6.5.1.4-3

4/6.5.1.4.2

Funktionen

```
13FE 00         FNNR:     .BY 0              ;NUMMER DER FUNKTION
                EEVAL:                       ;** ARITHM. ELEM.**
                ;           ;** AUSWERTUNG    **
13FF A9 00                LDA #0
1401 85 0F                STA VALTYP         ;TYPFLAG NUMERISCH
1403 20 80 03             JSR CHRGET         ;NAECHSTES ZEICHEN
1406 90 2D                BCC FNBAS          ;CODE WAR ZIFFER
1408 C9 C5                CMP #$C5           ;VAL-KODE
140A F0 68                BEQ FKTVAL
140C C9 60                CMP #$60           ;TEST AUF BUCHSTABE
140E B0 25                BCS FNBAS          ;
1410 C9 24                CMP #$24           ;"$".
1412 F0 6D                BEQ HEXCON         ;JA, DANN HEXZAHL
1414 C9 25                CMP #$25           ;"%"
1416 F0 76                BEQ BINCON         ;JA, DANN BINZAHL
1418 C9 41                CMP #$41           ;VERGLEICH MIT "A"
141A 90 18                BCC FNBASC         ;SONDERZEICHEN
141C 8D 6E 13             STA AKKUSICH       ;AKKU SICHERN
141F A2 00                LDX #0
1421 8E FE 13             STX FNNR           ;BEFNR =0
1424 8D 01 FF             STA SWRAM0         ;FUER BASIC-TEXT
                FE1:
1427 A0 00                LDY #0
1429 EE FE 13             INC FNNR           ;BEFNR=BEFNR+1
142C BD A3 14             LDA FTAB,X         ;ZEICH. AUS BEF.TAB.
142F D0 0D                BNE FE2            ;KEIN TRENNZEICHEN
1431 AD 6E 13             LDA AKKUSICH       ;AKKU RESTAURIEREN
                FNBASC:
1434 38                   SEC                ;CARRY WIEDER SETZEN
                FNBAS:
1435 8D 03 FF             STA SWROMRA0       ;STANDARD-KONFIG.
1438 20 65 13             JSR DFCRA7         ;BEFEHLSZEIGER RUECK
143B 4C DA 78             JMP NEVAL          ;STANDARD-ROUTINE
                FE2:
143E D1 3D                CMP (TXTPTR),Y     ;VERGL. M. BASIC-TEXT
1440 D0 28                BNE FE4            ;KEINE UEBEREINST.
1442 C8                   INY                ;NEXT ZEI. BASIC-TEXT
1443 E8                   INX                ;NEXT ZEI. BEF.TAB.
1444 BD A3 14             LDA FTAB,X         ;ZEICHEN AUS BEF.TAB.
1447 D0 F5                BNE FE2            ;NEXT ZEICH. PRUEFEN
1449 18                   CLC
144A 98                   TYA
144B 65 3D                ADC TXTPTR         ;BEFEHLSZEIGER UM
144D 85 3D                STA TXTPTR         ;BEF.-LAENGE ERHOEH.
144F 90 02                BCC FE3            ;KEIN UEBERTRAG
1451 E6 3E                INC TXTPTR+1       ;UEBERTRAG
                FE3:
1453 AD FE 13             LDA FNNR           ;BEFEHLSNUMMER HOLEN
1456 0A                   ASL                ;VERDOPPELN UND ALS
1457 AA                   TAX                ;ZEIGER IN SPRUNGTAB.
1458 CA                   DEX
```

Listing 4/6.5.1.4-4 (Teil 1)

```
1459 CA                          DEX
145A BD 9B 14                    LDA FSTAB,X
145D 8D F4 03                    STA ZIEL
1460 BD 9C 14                    LDA FSTAB+1,X
1463 8D F5 03                    STA ZIEL+1          ;SPRUNGZIEL
1466 20 F0 03                    JSR WEITSPR
1469 60                          RTS                 ;FERTIG
                    ;
                    FE4:
146A E8                          INX                 ;NEXT ZEI. BEF.TAB.
146B BD A3 14                    LDA FTAB,X
146E D0 FA                       BNE FE4             ;BEFEHL NICHT ZU ENDE
1470 E8                          INX                 ;'0' UEBERLESEN
1471 4C 27 14                    JMP FE1             ;NEXT BEFEHL CHECKEN
                    ;
                    FKTVAL:
1474 A9 09                       LDA #<VALNEU
1476 8D F4 03                    STA ZIEL
1479 A9 04                       LDA #>VALNEU
147B 8D F5 03                    STA ZIEL+1
147E 4C F0 03                    JMP WEITSPR
                    ;
                    HEXCON:
1481 A9 00                       LDA #<HEXCONV
1483 8D F4 03                    STA ZIEL
1486 A9 04                       LDA #>HEXCONV
1488 8D F5 03                    STA ZIEL+1
148B 4C F0 03                    JMP WEITSPR
                    ;
                    BINCON:
148E A9 03                       LDA #<BINCONV
1490 8D F4 03                    STA ZIEL
1493 A9 04                       LDA #>BINCONV
1495 8D F5 03                    STA ZIEL+1
1498 4C F0 03                    JMP WEITSPR
                    ;
                    ;
                    ;SPRUNGTABELLE FUNKTIONEN
                    ;
                    FSTAB:
149B 06 04                       .WO BINSTR          ;BIN$
149D 09 08                       .WO QEEK            ;QEEK
149F 0F 08                       .WO TIME            ;TIME
14A1 12 08                       .WO TEST            ;TEST-FUNKTION
                    ;
                    FTAB:
14A3 42 49 4E                    .BY "BIN$",0
24 00
14A8 51 45 45                    .BY "QEEK",0
4B 00
14AD 54 49 4D                    .BY "TIME",0
45 00
14B2 54 45 53                    .BY "TEST",0
54 00
14B7 00                          .BY 0               ;ENDE FUNKTIONSTAB.
                    ;
                    ;
```

Listing 4/6.5.1.4-4 (Teil 2)

Der Programmablauf für die Funktionsdekodierung ist weitgehend identisch mit dem für die Befehlsdekodierung. Für die Auswertung von hexadezimalen Operanden, von binären Operanden und die erweiterte VAL-Funktion wurden drei zusätzliche Abfragen am Anfang des Programms eingefügt. Im Gegensatz zu der Befehls-Routine endet diese Routine nicht mit JMP NEWSTT sondern mit RTS.

Es ist zu beachten, daß die Hilfszelle AKKUSICH, die Hilfsroutine DECBZ, der Vektor ZIEL und die Weitsprung-Routine auch von der Befehlsdekodier-Routine benützt werden. Man könnte meinen, daß sich dadurch Probleme ergeben. Jedoch zeigt sich, daß die Funktionsdekodier-Routine von der Befehlsdekodier-Routine nur indirekt über einen Befehl aufgerufen werden kann und das zu einem Zeitpunkt, wo die Befehlsdekodier-Routine bereits ihre Arbeit erledigt hat und deshalb diese Hilfszellen nicht mehr benötigt.

Auch hier gibt es eine korrespondierende Sprungtabelle in Bank 1, die auch einige wichtige Hilfsroutinen enthält:

```
                   ; *** SPRUNGLEISTE ***
0400 4C F2 04                JMP HEXCONV
0403 4C 14 05                JMP BINCONV
0406 4C 7B 05                JMP BINSTR
0409 4C 36 05                JMP VALNEU
                   ;
040C 4C 5D 04                JMP FLPNOS
040F 4C 72 04                JMP HEXINT
0412 4C AA 04                JMP BININT
0415 4C CB 04                JMP INTBIN
0418 4C E0 04                JMP INCTXT
041B 4C E7 04                JMP BZPLY
                   ;
```

Listing 4/6.5.1.4-5

4/6.5.1.5

Übersicht über die neuen Befehle

Selbstverständlich wollen wir Ihnen auch eine Zusammenfassung der neuen Befehle geben. Hier eine Aufstellung:

Befehl (*=Funktion)		Bedeutung	Adreßbereich in Bank 1
$hhhh	*	Erfassen Hexadezimalwert	$04F2 — $0513
%bbbbbbbbbbbbbbbb	*	Erfassen Binärwert	$0514 — $0535
AUS		Befehlserweiterung ausschalten	$0815 — $0839
BIN$(Adr)	*	Binärstring aus Adresse bilden	$057B — $0595
PAUSE HSEK		Pause	$0888 — $08D7
QEEK (Adr)	*	16-Bit-PEEK	$0861 — $0887
QOKE Adr, Adr		16-Bit-POKE	$083A — $0860
SETTIME U, ZE$		Alarmzeit im CIA setzen	$08D8 — $0980
TEST			$09FD — $09FD
TIME1	*	Uhrzeit aus CIA 1 lesen	$0981 — $09FC
TIME2	*	Uhrzeit aus CIA 2 lesen	
VAL(V$)	*	Zahlwert einer Zeichenreihe errechnen	$0536 — $057B

Hilfsroutinen sind nicht angeführt; die angeführten Parameter haben folgende Bedeutung:

b	—	Binärziffer
h	—	Hexadezimale Ziffer
Adr	—	Ganze Zahl (Adresse) zwischen 0 und 65535
HSEK	—	Zeit in 1/100 Sekunde
V$	—	Zeichenreihe aus Ziffern
U	—	Uhrnummer: 1 — Uhr im CIA 1
		2 — Uhr im CIA 2
		3 — Alarmzeit im CIA 1
		4 — Alarmzeit im CIA 2
ZE$	—	Zeit-Zeichenreihe im Format HH:MM:SS:Z

4/6.5.1.6

Parameterübergabe

Bereits bei den Zahlkonvertierungen haben wir verschiedene Parametertypen kennengelernt. Wir wollen sie hier noch einmal kurz zusammenstellen.

1. Numerische Parameter

Das Commodore-Basic rechnet grundsätzlich mit Fließkommazahlen, aber von der Verwendung her sind auch oft nur eingeschränkte Zahlenbereiche interessant. Folgende Möglichkeiten sind wichtig:

1.1 Fließkommazahlen
1.2 Byte-Parameter: Ganze Zahlen von 0 bis +255
1.3 Adreß-Parameter oder „non-signed integers“:
Ganze Zahlen von 0 bis +65535
1.4 „Signed-Integers“: Ganze Zahlen von -32768 bis +32767

2. Variablen

Eine Variable als Parameter zu übergeben heißt die Adresse der Variable dem Maschinenprogramm mitzuteilen. Dieses kann dann auf die Variable schreibend oder lesend zugreifen. Somit kann der Inhalt der Variablen als Ein- oder Ausgabeparameter verwendet werden.

3. Stringparameter

Ein Stringparameter bedeutet beim Commodore-Basic einen Deskriptor zu übergeben. Der Deskriptor beinhaltet die Länge der Strings und die Startadresse (Low Byte, High Byte) desselben.

4. Parameterübergabe an beliebige Maschinenprogramme

Die Parameter für Maschinenprogramme, die man einfach mit SYS aufrufen möchte, müssen ja nicht unbedingt dahinter im Befehlstext stehen. Man kann statt dessen folgende einfache Methode verwenden: Man sucht sich im Speicher einige freie Speicheradressen aus und schreibt in diese Zellen die Eingabeparameter. Das Maschinenprogramm kann nun genau diese Adressen benützen, da diese vorher fest vereinbart wurden. Vor dem Aufruf mit SYS muß man dann natürlich die Parameter einPOKEn.

Die Behandlung von Ausgabeparametern kann auf ähnlich einfache Weise geschehen; das Ergebnis kann man dann mit PEEK aus den spezifizierten Speicherzellen herauslesen. Wenn Sie anfangen, mit Maschinenprogrammen zu experimentieren, werden Sie vor allem diese Methode benützen, da Sie die Parameter natürlich auch im Monitor verändern können, und Sie daher immer eine gute Kontrolle über den Programmablauf haben.

Braucht man nur wenige Parameter, so kann man die Prozessorregister zur Übergabe verwenden. Der SYS-Befehl des Commodore 128 besitzt nämlich die praktische Eigenheit, daß unmittelbar vor dem Aufruf des eigentlichen Maschinenprogramms die Prozessorregister Status, Akku, X, Y aus den Speicherzellen 5 bis 8 (vergleiche im Handbuch Seite H7) geladen werden und nach Verlassen der Routine die geänderten Werte in diese Zellen zurückgeschrieben werden. Deshalb kann man eben diese Speicherzellen wieder als Ein- oder Ausgabeparameter verwenden.

Eingabeparameterübergabe aus dem Basictext

Die wichtigste RAM-Routine hierzu ist FRMEVL. Sie wertet einen Ausdruck beliebigen Typs und beliebiger Komplexität aus. Bei einem numerischen Parameter steht das Ergebnis im Fließkomma-Akkumulator, bei einem Stringparameter kann mit FRESTR die Stringadresse in die Zero-Page-Zellen INDEX, INDEX+1 geholt werden, die Länge steht dann im Akkumulator.

Außer FRMEVL gibt es noch viele andere Routinen, die eine Kombination von FRMEVL mit anderen Routinen darstellen. So z.B. GETBYT zum Holen eines Byteparameters in das X-Register.

Eine Kurzdarstellung der wichtigsten Parameterhol-Routinen geben wir anschließend, Beispiele finden Sie auch noch bei den einzelnen Befehlen und Funktionen.

Routine	Bedeutung	Beispiel	
CHKCOM	prüfen auf Komma	QOKE	$1083D
CHKNUM	numerische Prüfung	BIN$TR	$1057E
CHRGET	holt Zeichen aus Basic-Text	EEVAL	$013D5
FRESTR	holt Stringparameter	SETTIME	$108E8
FRMEVL	holt beliebigen Parameter	PAUSE	$1088D
GETBYT	holt numerischen Wert (0-255)	SETTIME	$108DB
GETNUM	holt numerischen Wert (Fließk.)		
GETWRD	holt numerischen Wert (0-65535)	QOKE	$1083A
LEN1	FRESTR+Typflag prüfen	VALNEU	$1053C
PARCHK	holt Ausdruck in Klammern	BIN$TR	$1057B

Funktionen, die einen numerischen Wert liefern sollen

Diese Funktionen müssen das Ergebnis als Fließkommazahl im Fließkomma-Akkumulator zurückgeben. Es muß darauf geachtet werden, daß das Flag für den Parametertyp VALTYP auf numerisch (=0) gesetzt wird.

Funktionen mit Strings als Ausgabeparameter

Im folgenden werden zwei wichtige Möglichkeiten zur Lösung dieser Aufgabe besprochen. Das ist zum einen die Verwendung der Routinen GETSPA und PUTNEW, zum anderen das Ablegen der Strings im Pufferbereich ab $00FF und anschließende Verwendung der Routine STRLIT.

Die erste Möglichkeit ist z.B. bei der Routine zum Uhrzeit-Lesen verwendet worden. Die Routine GETSPA schafft Platz für einen neuen String, dessen Länge im Akkumulator übergeben werden muß. Die Routine übergibt im Akkumulator nochmals die Länge des Strings und im X- und Y-Register das Low- und Highbyte der Stringadresse, wo sich der neue String nun befindet. Zweckmäßigerweise speichert man diese drei Byte nacheinander im Fließkomma-Akkumulator ab und beschreibt dann den String. Anschließend ruft man die Routine PUTNEW auf, die den Deskriptor aus dem Fließkomma-Akkumulator auf den Stringstapel überträgt. Der zuletzt auf den Stringstapel gelegte String ist das Ergebnis der Routine.

Die andere Möglichkeit geht davon aus, daß der String ab $00FF beginnt und durch ein 0-Byte begrenzt wird. Dann genügt es, die Routine STRLIT aufzurufen, die dann alles weitere erledigt.

4/6.5.2

Zahlenumwandlung

In diesem Kapitel sind die Routinen zur Auswertung von hexadezimalen und binären Parametern aus dem Basic-Text, die erweiterte VAL-Funktion und die BIN$-Funktion sowie einige dazugehörige Hilfsroutinen zusammengefaßt.

Symbolvereinbarungen

```
PUFFER:     .EQ $0100       ;FUER STRINGBILDUNG
H1:         .EQ FACEXP      ;1.HILFSZELLE
H2:         .EQ H1+1        ;2.HILFSZELLE
H3:         .EQ H2+1        ;3.HILFSZELLE
LEN1:       .EQ $866E       ;FRESTR / VALTYP
;
```

Listing 4/6.5.2-1

Das Symbol PUFFER beschreibt den Zwischenspeicher zum Aufbau einer Zeichenreihe, wie bei den Zahlkonvertierungs-Routinen in Kapitel 4/6.3.2 beschrieben. Die Hilfszellen H1 bis H3 begegneten uns ebenfalls dort. Die Basic-Routine LEN1 ist nichts anderes als ein Aufruf von FRESTR, der zusätzlich noch das Typflag auf numerisch setzt.

Sprungleiste

```
                 ; *** SPRUNGLEISTE ***
0400 4C F2 04                 JMP HEXCONV
0403 4C 14 05                 JMP BINCONV
0406 4C 7B 05                 JMP BINSTR
0409 4C 36 05                 JMP VALNEU
                 ;
040C 4C 5D 04                 JMP FLPNOS
040F 4C 72 04                 JMP HEXINT
0412 4C AA 04                 JMP BININT
0415 4C CB 04                 JMP INTBIN
0418 4C E0 04                 JMP INCTXT
041B 4C E7 04                 JMP BZPLY
                 ;
```

Listing 4/6.5.2-2

In dieser Sprungleiste sind die Labels für die vier neuen Basic-Funktionen sowie die wichtigsten Hilfsroutinen dieses Abschnitts zusammengefaßt.

Hilfsroutinen zum Erhöhen des Basic-Befehlszeigers

```
                INCTXT:                              ;TXTPTR UM 1 ERHOEHEN
04E0 E6 3D                   INC TXTPTR
04E2 D0 02                   BNE INCTXT1
04E4 E6 3E                   INC TXTPTR+1            ;UEBERTRAG
                INCTXT1:
04E6 60                      RTS
                ;
```

Listing 4/6.5.2-3

Offensichtlich macht diese Routine nichts anderes als den Zeiger im Zellenpaar TXTPTR, TXTPTR+1 um eins zu erhöhen.

Basic-Befehlsanzeiger um Y erhöhen

```
               BZPLY:                          ;TXTPTR UM Y ERHOEHEN
04E7 98                    TYA
04E8 18                    CLC
04E9 65 3D                 ADC TXTPTR          ;ADDITION
04EB 85 3D                 STA TXTPTR
04ED 90 02                 BCC BZPLY1          ;KEIN UEBERTRAG
04EF E6 3E                 INC TXTPTR+1        ;HIGH-BYTE ERHOEHEN
               BZPLY1:
04F1 60                    RTS                 ;FERTIG
               ;
```

Listing 4/6.5.2-4

Diese Routine macht etwas ähnliches wie die obenstehende, jedoch kann man hier im Y-Register die Anzahl der Zeichen angeben, um die der Befehlszeiger weitergestellt wird.

Hexadezimalen Operaden auswerten

```
               HEXCONV:
04F2 20 E0 04              JSR INCTXT          ;1. ZEICHEN ANPEILEN
04F5 A0 04                 LDY #4              ;5 BYTE KOPIEREN
               HEXC1:
04F7 A9 3D                 LDA #TXTPTR
04F9 A2 00                 LDX #0
04FB 20 74 FF              JSR INDFET
04FE 99 00 01              STA PUFFER,Y        ;IN PUFFERBERICH
0501 88                    DEY
0502 10 F3                 BPL HEXC1
0504 A0 04                 LDY #4              ;MAX. 4 HEXZIFFERN
0506 20 72 04              JSR HEXINT          ;UMWANDLUNG
0509 20 E7 04              JSR BZPLY           ;BAS.BZ. UM Y ERH.
050C A0 00                 LDY #0
050E 84 0F                 STY VALTYP          ;TYPFLAG NUMERISCH
0510 20 62 05              JSR INTFLP          ;ERGEBNIS -> FAC
0513 60                    RTS                 ;FERTIG
               ;
```

Listing 4/6.5.2-5

Mit der ersten Programmzeile wird die erste hexadezimale Ziffer angepeilt. Daraufhin werden fünf Zeichen aus dem Basictext in den Pufferbereich übertragen, wobei die Kernal-Routine INDFET verwendet wird, die das Bank-Switching managt.

Anschließend wird die Umwandlungs-Routine HEXINT aufgerufen, der Basic-Befehlszeiger korrigiert und das Ergebnis nach Fließkomma umgewandelt.

Binären Operanden auswerten

```
                  BINCONV:
0514 20 E0 04               JSR INCTXT        ;1. ZEICHEN ANPEILEN
0517 A0 10                  LDY #16           ;17 BYTE KOPIEREN
                  BINC1:
0519 A9 3D                  LDA #TXTPTR
051B A2 00                  LDX #0
051D 20 74 FF               JSR INDFET
0520 99 00 01               STA PUFFER,Y
0523 88                     DEY
0524 10 F3                  BPL BINC1
0526 A0 10                  LDY #16           ;MAX. 16 BINZIFFERN
0528 20 AA 04               JSR BININT        ;UMWANDLUNG
052B 20 E7 04               JSR BZPLY         ;BAS.BZ. UM Y ERH.
052E A0 00                  LDY #0
0530 84 0F                  STY VALTYP        ;TYPFLAG NUMERISCH
0532 20 62 05               JSR INTFLP        ;ERGEBNIS -> FAC
0535 60                     RTS               ;FERTIG
                  ;
```

Listing 4/6.5.2-6

Der Programmablauf ist völlig analog zum Auswerten eines hexadezimalen Operanden; lediglich werden hier 17 Byte in den Pufferbereich kopiert.

Die Routine INTFLP wird im nächsten Kapitel erläutert.

Erweiterte VAL-Funktionen

```
                  VALNEU:
0536 20 E0 04               JSR INCTXT        ;VAL-CODE UEBERLESEN
0539 20 27 04               JSR EPARCHK       ;WERT IN () HOLEN
053C 20 6E 86               JSR LEN1          ;FRESTR / TYPFLAG
053F 85 63                  STA H1            ;STRINGLAENGE
0541 A0 00                  LDY #0
0543 20 B7 03               JSR INDINIR1      ;ERSTES ZEICHEN
0546 C9 24                  CMP #$24          ;"$"
0548 F0 1F                  BEQ HEXVAL        ;VAL VON HEXZAHL
054A C9 25                  CMP #$25          ;"%"
054C F0 24                  BEQ BINVAL        ;VAL VON BINAERZAHL
054E A5 63                  LDA H1            ;STRINGLAENGE
0550 4C 4D 80               JMP VAL+3         ;ZUR VAL-FUNKTION
                  VAL:      .EQ $804A         ;
                  ;
                  VALPAR:                     ;HILFSROUTINE
0553 A4 63                  LDY H1            ;STRINGLAENGE
                  VALPAR1:
0555 20 B7 03               JSR INDINIR1      ;RAM1 AUS RAM1 LADEN
0558 88                     DEY
0559 99 00 01               STA PUFFER,Y      ;IN PUFFER
055C D0 F7                  BNE VALPAR1
055E A4 63                  LDY H1
0560 88                     DEY
0561 60                     RTS
```

Listing 4/6.5.2-7 (Teil 1)

```
                  ;
                  INTFLP:
0562 A5 64                  LDA H2
0564 A4 65                  LDY H3
0566 4C C9 84               JMP NOSFLT
                  ;
                  HEXVAL:
0569 20 53 05               JSR VALPAR      ;STRING UEBERTRAGEN
056C 20 72 04               JSR HEXINT      ;UMWANDLUNG
056F 4C 62 05               JMP INTFLP      ;ERGEBNIS -> FAC
                  ;
                  BINVAL:
0572 20 53 05               JSR VALPAR      ;STRING UEBERTRAGEN
0575 20 AA 04               JSR BININT      ;UMWANDLUNG
0578 4C 62 05               JMP INTFLP      ;ERGEBNIS -> FAC
                  ;
```

Listing 4/6.5.2-7 (Teil 2)

Im ersten Teil des Listings spricht dieses wohl für sich selbst, es wird lediglich in Abhängigkeit vom ersten Zeichen des Strings zu HEXVAL, zu BINVAL bzw. in die normale VAL-Routine verzweigt.

Der nächste Abschnitt ab VALPAR ist eine Hilfsroutine, die den String in den Pufferbereich kopiert. Die folgende kleine Routine INTFLP lädt Akku und Y-Register aus den Hilfszellen H2 und H3, damit die Parameterkonventionen für die Routine NOSFLT eingehalten werden.

Die letzten beiden Programmstücke ab HEXVAL und BINVAL sprechen wieder für sich selbst.

BIN$-Funktion

```
                  BINSTR:
057B 20 27 04               JSR EPARCHK     ;AUSDRUCK IN () HOLEN
057E 20 DA 77               JSR CHKNUM      ;NUMERISCH PRUEFEN
0581 20 5D 04               JSR FLPNOS      ;-> INTEGER
0584 85 64                  STA H2
0586 84 65                  STY H3
0588 A0 25                  LDY #$25        ;"%"
058A 84 FF                  STY PUFFER-1    ;ERSTES ZEICHEN
058C A0 10                  LDY #16         ;16 STELLEN
058E 20 C8 04               JSR INTBIN      ;UMWANDLUNG
0591 A9 FF                  LDA #$FF
0593 A0 00                  LDY #$00        ;STRINGADR.=$00FF
0595 4C 9A 86               JMP STRLIT      ;WEITER WIE STR$
                  ;
```

Listing 4/6.5.2-8

Hier wird zunächst der Parameter ausgewertet und in die Hilfszellen H2 und H3 gelegt. Dann wird das erste Zeichen des Strings, den wir bei $00FF beginnen lassen wollen, mit einem Prozentzeichen besetzt. Anschließend wird die Umwandlungs-

Routine INTBIN, die in Kapitel 4/6.3.2 besprochen wurde, aufgerufen und anschließend der String mit Hilfe von STRLIT in den eigentlichen Stringbereich transferiert und der Deskriptor des neuen Strings auf den Stringstapel gelegt.

4/6.5.3

Uhrzeit und Pause

Die Zeitbehandlung beim Commodore 128 kann auf verschiedene Weisen erfolgen. Zum einen kann man natürlich die eingebaute TI/TI$-Uhr, die sogenannte Jiffy-Clock, verwenden; genauer ist aber die Verwendung der Echtzeituhren oder der Timer der CIA's (siehe Kapitel 2/2.1.5). Die Timer laufen quarzgenau, und die Echtzeituhr läuft netzfrequenzgetrieben und ist daher über lange Zeiten noch genauer, auch wenn kurzfristig Abweichungen bis zu drei Minuten auftreten können.

Die Timer im CIA können maximal mit einem 32-Bit-Wert geladen werden, der dann mit 1 MHz heruntergezählt wird, so daß auf diese Weise nur Zeiten bis etwa einer Stunde realisierbar sind. Wenigstens theoretisch liegt aber die Genauigkeit dieser Uhr bei einer Mikrosekunde.

Ganz anders ist die Echtzeituhr aufgebaut. Sie ist wie eine Digitaluhr mit Stunden-, Minuten-, Sekunden- und Zehntelsekunden-Register ausgerüstet und verfügt außerdem noch über ein Vormittags-, Nachmittags-Bit (AM/PM).

In diesem Abschnitt wollen wir beide Möglichkeiten ausnützen und zwar werden wir zunächst eine Pause mit den Timern realisieren, die auf 1/100 Sekunde genau angegeben und eingehalten werden kann (der SLEEP-Befehl funktioniert nur ab einer Sekunde aufwärts). Als zweites werden wir Routinen vorstellen, die die Uhren der CIA's stellen und lesen können.

4/6.5.3.1

Pause

```
                ;******************************
                ;* PAUSE BIS ZU 655.35 SEC   *
                ;******************************
                FK100:                                 ;FLIESSK.KONSTANTE 100
0888 87 48 00             .BY $87,$48,$00,$00,$00
00 00
                ;
                CIA2:     .EQ $DD00                    ;ADRESSE DES CIA2
                FZ1  :    .EQ $FA                      ;AB HIER FREIE ZERO-PAGE
                ;
                PAUSE:
088D 20 30 04             JSR EFRMEVL                  ;ZEITPARAMETER HOLEN
0890 20 DA 77             JSR CHKNUM
0893 A9 88                LDA #<FK100
0895 A0 08                LDY #>FK100
0897 20 24 8A             JSR FMULT                    ;MAL KONSTANTE 100
089A 20 5D 04             JSR FLPNOS                   ;-> INTEGER -> A,Y
089D 48                   PHA                          ;MSB SICHERN
089E 98                   TYA
089F 48                   PHA                          ;LSB SICHERN
                ;
08A0 A9 40                LDA #RA1ROMIO
08A2 8D 00 FF             STA MMUCR
08A5 A9 7A                LDA #<9850                   ;TEILRATE A LOW-BYTE
08A7 8D 04 DD             STA CIA2+4
08AA A9 26                LDA #>9850                   ;TEILRATE A HIGH-BYTE
08AC 8D 05 DD             STA CIA2+5
08AF 68                   PLA
08B0 8D 06 DD             STA CIA2+6                   ;TEILRATE B LOW -BYTE
08B3 68                   PLA
08B4 8D 07 DD             STA CIA2+7                   ;TEILRATE B HIGH-BYTE
08B7 AD 0E DD             LDA CIA2+14                  ;KONTROLLREG. A
08BA 29 D1                AND #%11010001
08BC 09 11                ORA #%00010001
08BE 8D 0E DD             STA CIA2+14
08C1 AD 0F DD             LDA CIA2+15                  ;KONTROLLREG. B
08C4 29 D1                AND #%11010001
08C6 09 59                ORA #%01011001
08C8 8D 0F DD             STA CIA2+15
                PAULOOP:
08CB AD 0D DD             LDA CIA2+13                  ;INTERRUPT-FLAG-REG.
08CE 29 02                AND #%00000010
08D0 D0 05                BNE PAUEND
08D2 20 E1 FF             JSR STOP                     ;STOPTASTE ABFRAGEN
08D5 D0 F4                BNE PAULOOP                  ;SCHLEIFE
                PAUEND:
08D7 60                   RTS                          ;ENDE PAUSE
```

Listing 4/6.5.3-1

Zunächst werden einige Besonderheiten des Programms angemerkt. Wir verwenden die Timer A und B des CIA2, die normalerweise nicht gebraucht werden, es sei denn für RS-232 Ein-/Ausgabe. Wir benötigen ein Hilfszellenpaar in der Zero-Page, wofür wir zwei Byte aus dem freien Bereich $FA bis $FF benützen. Da die Zeit auf 1/100 Sekunde genau angegeben werden kann, werden wir den Zeitparameter zunächst mit 100 multiplizieren und dann eine ganze Zahl bilden, welche dann die Anzahl der hundertstel Sekunden darstellt.

Wie immer bei selbstgemachten Assembler-Routinen für den 128 treten auch an dieser Stelle Bank-Switch-Probleme auf, die jedoch hier recht einfach gelöst werden. Zum einen geschieht das Holen des Parameters mit Hilfe der Routine EFRMEVL, wodurch die Rückkehr in unser Programm gesichert wird, zum anderen wird die MMU mit einer Konfiguration geladen, die auch den I/O-Bereich enthält. Dadurch kann überhaupt der CIA erst beschrieben werden.

Zum Programmablauf: Nach dem Holen des Parameters und Multiplikation mit 100 wird die Zahl mit Hilfe von FLPNOS (siehe Kapitel 4/6.5.3.2.2) in eine vorzeichenlose ganze Zahl umgewandelt, die wir zunächst auf den Stapelspeicher legen. Danach wird die MMU umkonfiguriert und die Teilrate für den Timer A auf 9850 gesetzt, wodurch der Timer B mit 100 Hertz getriggert wird. Dadurch können wir unseren umgerechneten Parameter als Teilrate für den Timer B einschreiben.

Die Timer werden auf verkettete Betriebsart geschaltet, und dann wird in einer Schleife gewartet, bis das entsprechende Interrupt-Flag anzeigt, daß Timer B abgelaufen ist. Zwischendurch wird immer auch noch die Stop-Taste abgefragt, damit man im Zweifelsfalle eine zu lange Pause unterbrechen kann.

4/6.5.3.2

SETTIME — Uhrzeit im CIA setzen

Wenn der 128er schon über gleich zwei Echtzeituhren verfügt, so sollte man doch auch einen Befehl haben, um diese in bequemer Weise zu stellen. Da die CIA's auch über ein Alarmzeit-Register verfügen, wurde dieser Befehl so gestaltet, daß durch Angabe eines Parameters auch die Alarmzeiten gesetzt werden können.

Die Behandlung des Alarmfalles, wenn Uhrzeit und Alarmzeit übereinstimmen, wird hier nicht gezeigt. Im Prinzip müßte man dabei so vorgehen, daß man im

CIA-Interrupt-Control-Register das einschlägige Bit setzt, wodurch dann im Falle eines Falles entweder ein IRQ (bei CIA1) oder ein NMI (bei CIA2) ausgelöst wird. Die Interupt-Routine muß dann abfragen, ob der Interrupt wirklich vom Alarm eines CIA gekommen ist.

Wir wollen folgende Syntax vereinbaren:

SETTIME UHRNUMMER, ZEIT

Dabei bedeuten:

UHRNUMMER eine Zahl zwischen 1 und 4:

1	=	Uhr in CIA 1
2	=	Uhr in CIA 2
3	=	Alarmzeit in CIA 1
4	=	Alarmzeit in CIA 2

ZEIT = Zeichenreihe aus 10 Zeichen in folgendem Format: „HH:MM:SS:Z“. Die Zeit wird im 24-Stunden-Format angegeben.

```
                   ;******************************
                   ;* UHRZEIT IM CIA SETZEN      *
                   ;******************************
08D8 00            CLOCKNR:    .BY 0
08D9 00            ZAEHL:      .BY 0
08DA 00            HILF:       .BY 0
                   ;
                   SETTIME:
08DB 20 39 04                  JSR EGETBYT      ;UHRNUMMER HOLEN
08DE CA                        DEX              ;EINS ABZIEHEN
08DF 8E D8 08                  STX CLOCKNR      ;CLOCKNR. (0,1,2,3)
08E2 20 1E 04                  JSR ECHKCOM      ;KOMMA
08E5 20 30 04                  JSR EFRMEVL      ;STRING HOLEN
08E8 20 7E 87                  JSR FRESTR
08EB C9 0A                     CMP #10          ;10 ZEICHEN
08ED F0 03                     BEQ SETTIM1      ;OK
08EF 4C 28 7D                  JMP FCERR        ;FEHLER
                   SETTIM1:
08F2 A9 00                     LDA #0
08F4 8D D9 08                  STA ZAEHL
08F7 4E D8 08                  LSR CLOCKNR      ;0-BIT IN CARRY
08FA A9 00                     LDA #0
08FC 85 FA                     STA FZ1          ;ZEIGER AUF CIA
08FE 69 DC                     ADC #$DC         ;HIGH-BYTE VON CIA
0900 85 FB                     STA FZ1+1
0902 A9 40                     LDA #RA1ROMIO
0904 8D 00 FF                  STA MMUCR
0907 A0 0E                     LDY #14
0909 B1 FA                     LDA (FZ1),Y      ;KONTROLLREG. A
090B 09 80                     ORA #%10000000   ;50HZ-FLAG SETZEN
090D 91 FA                     STA (FZ1),Y
```

Listing 4/6.5.3-2 (Teil 1)

```
090F C8                         INY
0910 AD D8 08                   LDA CLOCKNR          ;CLOCKNR. / 2
0913 4A                         LSR                  ;0-BIT INS CARRY
0914 08                         PHP                  ;CARRY SICHERN
0915 B1 FA                      LDA (FZ1),Y          ;KONTROLLREG. B
0917 2A                         ROL
0918 28                         PLP                  ;CARRY VON OBEN
0919 6A                         ROR
091A 91 FA                      STA (FZ1),Y          ;ALARMZEIT/UHRZEIT
091C 20 4E 09                   JSR AUSW2            ;2 ZIFFERN AUSW.
091F F8                         SED                  ;BCD-ARITHMETIK
0920 C9 12                      CMP #$12             ;12 UHR
0922 F0 06                      BEQ SETT2            ;PM-FLAG AUTOM.
0924 90 04                      BCC SETT2            ;VOR 12 UHR
0926 E9 12                      SBC #$12             ;12 ABZIEHEN
                  SETTPM:
0928 09 80                      ORA #%10000000       ;PM-FLAG
                  SETT2:
092A D8                         CLD                  ;BCD AUS
092B A0 0B                      LDY #11
092D 91 FA                      STA (FZ1),Y          ;STUNDEN SETZEN
092F 20 4E 09                   JSR AUSW2            ;2 ZIFFERN AUSW.
0932 A0 0A                      LDY #10
0934 91 FA                      STA (FZ1),Y          ;MINUTEN SETZEN
0936 20 4E 09                   JSR AUSW2            ;2 ZIFFERN AUSW.
0939 A0 09                      LDY #9
093B 91 FA                      STA (FZ1),Y          ;SEKUNDEN SETZEN
093D 8D 02 FF                   STA SWRAM1           ;Y STEHT RICHTIG
0940 B1 24                      LDA (INDEX),Y        ;1/10 SEKUNDEN
0942 29 0F                      AND #$0F
0944 A2 40                      LDX #RA1ROMIO
0946 8E 00 FF                   STX MMUCR
0949 A0 08                      LDY #8
094B 91 FA                      STA (FZ1),Y          ;SETZEN
094D 60                         RTS
                  ;
                  AUSW2:                             ;2 ZIFFERN UND ":"
094E 8D 02 FF                   STA SWRAM1
0951 AC D9 08                   LDY ZAEHL
0954 B1 24                      LDA (INDEX),Y        ;NAECHSTES ZEICHEN
0956 0A                         ASL                  ;4 BIT NACH LINKS
0957 0A                         ASL
0958 0A                         ASL
0959 0A                         ASL
095A 8D DA 08                   STA HILF             ;MERKEN
095D C8                         INY
095E B1 24                      LDA (INDEX),Y        ;NAECHSTES ZEICHEN
0960 29 0F                      AND #$0F             ;4 BIT AUSBLENDEN
0962 0D DA 08                   ORA HILF             ;MIT 1.ZIFFER ODERN
0965 8D DA 08                   STA HILF             ;MERKEN
0968 C8                         INY
0969 B1 24                      LDA (INDEX),Y        ;NAECHSTES ZEICHEN
096B C9 3A                      CMP #$3A             ;":"
096D F0 05                      BEQ AUSW21           ;OK
096F 68                         PLA                  ;STACK BEREINIGEN
0970 68                         PLA
0971 4C 28 7D                   JMP FCERR            ;FEHLER
                  AUSW21:
0974 A9 40                      LDA #RA1ROMIO
0976 8D 00 FF                   STA MMUCR
```

Listing 4/6.5.3-2 (Teil 2)

```
0979 AD DA 08 665          LDA HILF          ;ERGEBNIS IN AKKU
097C C8       666          INY
097D 8C D9 08 667          STY ZAEHL         ;ZAEHLER MERKEN
0980 60       668          RTS               ;ENDE UPRO AUSW2
              669    ;
```

Listing 4/6.5.3-2 (Teil 3)

An Bank-Switch-Probleme haben wir uns jetzt ja schon gewöhnt, und hier sind im wesentlichen die gleichen zu bewältigen, wie beim oben beschriebenen Pause-Befehl, jedoch müssen wir hier noch einiges öfter hin- und herschalten. Der Programmablauf beginnt wieder mit dem Holen der Parameter, wobei der numerische Parameter mit EGETBYT und der String-Parameter mit EFRMEVL und FRESTR geholt wird. Als Hilfszellen benötigen wir CLOCKNR zur Aufnahme des Uhrnummern-Parameters, ZAEHL als Zeiger in den String und HILF zum Zwischenspeichern des Akkumulators.

In der Hilfszelle CLOCKNR wird die um eins verminderte Uhrnummer gespeichert, so daß wir den CIA1 ansprechen müssen, wenn der Inhalt dieser Zelle gerade ist, sonst CIA2. Wir werten die Informationen aus, indem wir das niederwertigste Bit dieser Zelle in das Carry schieben und dann eine Addition zum Wert $DC ausführen, womit wir im Akkumulator das höherwertige Byte der Basisadresse des richtigen CIA erhalten.

Einen Zeiger auf diesen CIA errichten wir in dem Hilfszellenpaar FZ1, FZ1+1. Auf noch trickreichere Weise, wie gerade eben, setzen oder löschen wir das Alarmzeit-/Uhrzeit-Bit des Kontroll-Registers B. Dazu schieben wir das nächste Bit der Uhrnummer in das Carry, sichern das Carry, laden nun das Kontroll-Register, rotieren es nach links, holen das alte Carry zurück und rotieren das Kontroll-Register erneut nach rechts und speichern es wieder ab. Auf diese Weise ist das Bit aus der Zelle CLOCKNR in das Kontroll-Register des CIA gewandert.

Wir müssen unsere Trickkiste nun noch einmal bemühen, denn die Uhrzeit muß ja noch vom 24-Stunden- in das 12-Stunden-Format umgerechnet werden. Dazu holen wir zunächst mit der Hilfsroutine AUSW2 die Stunden im BCD-Format in den Akku. Dann schalten wir mit dem SED-Befehl auch den Prozessor auf BCD-Arithmetik um, womit wir auf einfache Weise 12 abziehen können, was ja notwendig wird, wenn unsere Stundenzahl größer als 12 ist. In diesem Fall müssen wir natürlich noch das Nachmittags-Flag setzen, was ab dem Label SETTPM erledigt wird.

Im restlichen Programmstück werden lediglich die weiteren Ziffern gelesen und in die entsprechenden Uhrzeit-Register übertragen.

4/6.5.3.3

Uhrzeit aus Echtzeituhr lesen

Fast noch wichtiger als das Setzen der Uhren mit einem Basic-Befehl ist das Lesen der Uhr mit einer geeigneten Funktion, weil sonst das Lesen der Uhr länger dauert als das Genauigkeitsintervall der Uhr selbst.

```
                ;****************************
                ;* UHRZEIT IM CIA LESEN      *
                ;****************************
                TIME:
0981 20 39 04                  JSR EGETBYT        ;UHRNUMMER HOLEN
0984 8A                        TXA
0985 18                        CLC
0986 69 DB                     ADC #$DC-1         ;UHRNUMMER-1+$DC
0988 85 FB                     STA FZ1+1          ;FZ1 AUF CIA-BASIS
098A A9 00                     LDA #0
098C 85 FA                     STA FZ1
098E 8D D9 08                  STA ZAEHL          ;ZAEHLER AUF 0
0991 A9 0A                     LDA #10            ;10 ZEICHEN
0993 20 99 92                  JSR GETSPA         ;STRING EINRICHTEN
0996 85 63                     STA FACEXP         ;STRINGLAENGE
0998 86 64                     STX FACEXP+1       ;STRINGADR. LOW
099A 84 65                     STY FACEXP+2       ;STRINGADR. HIGH
099C A9 40                     LDA #RA1ROMIO
099E 8D 00 FF                  STA MMUCR
09A1 A0 0B                     LDY #11
09A3 B1 FA                     LDA (FZ1),Y        ;STUNDEN
09A5 F8                        SED                ;BCD-ARITHMETIK
09A6 08                        PHP                ;FLAG RETTEN
09A7 29 7F                     AND #%01111111     ;OHNE PM-FLAG
09A9 C9 12                     CMP #$12           ;12 UHR
09AB D0 02                     BNE TIME3
09AD A9 00                     LDA #$00           ;12 UHR AM = 0 UHR
                TIME3:
09AF 28                        PLP                ;FLAG HOLEN
09B0 10 03                     BPL TIME4          ;AM
09B2 18                        CLC
09B3 69 12                     ADC #$12           ;+ 12 STUNDEN
                TIME4:
09B5 D8                        CLD                ;BCD AUS
09B6 20 EF 09                  JSR TIMEUD         ;-> STRING
09B9 A0 0A                     LDY #10
09BB B1 FA                     LDA (FZ1),Y        ;MINUTEN
09BD 20 EF 09                  JSR TIMEUD         ;-> STRING
09C0 A0 09                     LDY #9
09C2 B1 FA                     LDA (FZ1),Y        ;SEKUNDEN
09C4 20 EF 09                  JSR TIMEUD         ;-> STRING
09C7 A0 08                     LDY #8
09C9 B1 FA                     LDA (FZ1),Y        ;1/10 SEKUNDEN
```

Listing 4/6.5.3-3 (Teil 1)

```
09CB 20 D8 09          JSR TIMEU2      ;-> STRING
09CE 8D 04 FF          STA SWROMRA1
09D1 4C E3 86          JMP PUTNEW      ;STRING AUF STAPEL
                   ;
              TIMEU1:                  ;HIGH-NYBBLE -> STRING
09D4 4A                LSR             ;4 BIT NACH RECHTS
09D5 4A                LSR
09D6 4A                LSR
09D7 4A                LSR
              TIMEU2:                  ;LOW-NYBBLE -> STRING
09D8 29 0F             AND #$0F        ;4 BIT MASKIEREN
09DA 09 30             ORA #$30        ;ASC-BITS FUER ZIFFER
              TIMEU3:                  ;AKKU -> STRING
09DC 48                PHA
09DD A9 64             LDA #FACEXP+1   ;ZEIGER AUF STRING
09DF 8D B9 02          STA STAVEC
09E2 68                PLA
09E3 AC D9 08          LDY ZAEHL       ;ZAEHLER IM STRING
09E6 A2 01             LDX #1          ;BANK 1
09E8 20 77 FF          JSR INDSTA      ;AKKU EINTRAGEN
09EB EE D9 08          INC ZAEHL       ;ZAEHLER ERHOEHEN
09EE 60                RTS
                   ;
              TIMEUD:                  ;BYTE EINTRAGEN
09EF 48                PHA             ;MERKEN
09F0 20 D4 09          JSR TIMEU1      ;HIGH-NYBBLE
09F3 68                PLA
09F4 20 D8 09          JSR TIMEU2      ;LOW-NYBBLE
09F7 A9 3A             LDA #$3A        ;":"
09F9 20 DC 09          JSR TIMEU3      ;EINTRAGEN
09FC 60                RTS
                   ;
```

Listing 4/6.5.3-3 (Teil 2)

Die Uhrzeit kann durch diese Funktion mit PRINT TIME1 bzw. PRINT TIME2 abgerufen werden, je nachdem, ob die Uhr aus CIA1 oder CIA2 gemeint ist. Unser Programm muß also mit dem Holen der Uhrnummer beginnen.

Ähnlich wie im vorigen Kapitel wird in den Zellen FZ1, FZ1+1 ein Zeiger auf die Basisadresse des entsprechenden CIA's errichtet. Dann wird mit Hilfe der Routine NEUSTR ein neuer String mit der Länge 10 eingerichtet und dessen Deskriptor im Fließkomma-Akkumulator zwischengespeichert, der für diese Zwecke mißbraucht wird. Die Stunden werden ebenfalls wieder mit dem Trick über die BCD-Arithmetik in 24-Stunden-Format umgerechnet, die anderen Ziffern sind dann wieder einfacher zu behandeln. Unser Programm hört auf mit einem Sprung auf die Routine PUTNEW, die den String, dessen Deskriptor im Fließkomma-Akkumulator steht, auf den Stringstapel legt. Dies ist die Vorgehensweise, wie man allgemein einen String als Ergebnisparameter bereitstellt.

Nun werfen wir noch einen Blick auf die Hilfsroutinen TIMEUD und TIMEU1 bis TIMEU3. Die Routine TIMEUD trägt zwei Ziffern und einen Doppelpunkt in den

String ab der momentanen Zählerstelle ein. Als Zähler verwenden wir übrigens die Hilfszelle ZAEHL aus dem vorigen Kapitel. Man sollte sehr vorsichtig sein, eine Hilfszelle für zwei verschiedene Unterprogramme zu verwenden, zumal hier ja nicht sichergestellt ist, daß man nicht vielleicht die Uhr des einen CIA's mit dem Stand des anderen setzen möchte, so daß die Routinen dann verschachtelt aufgerufen werden, wodurch unter Umständen die Hilfszelle verändert wird. In unserem konkreten Fall ist das jedoch durch die spezielle Form des Programmablaufs ausgeschlossen.

Die Routine TIMEU1 wandelt die oberen vier Bits des Akkumulators in eine Ziffer um und trägt sie in den String ein, TIMEU2 macht dies mit den niedrigeren vier Bit und TIMEU3 trägt nur den Inhalt des Akkumulators in den String ein. Letztere Routine wird dann benutzt um einen Doppelpunkt einzutragen.

4/6.5.4

16-Bit-Speicherbefehle

Der Basic-Interpreter und das Betriebssystem benutzen oft Speicherzellenpaare zur Speicherung einer vorzeichenlosen ganzen Zahl — etwa einer Adresse — wobei in der ersten der beiden Zellen das niederwertige Byte und in der zweiten das höherwertige Byte abgelegt wird. Wenn man solch ein Paar lesen oder schreiben möchte, so ist das mit den normalen PEEK- bzw. POKE-Befehlen nur umständlich möglich. Mit den hier vorgestellten Erweiterungen ist das kein Problem mehr. Der Zugriff auf jede beliebige Bank wird, wie bei den Standard-PEEK- bzw. POKE-Befehlen, mit dem BANK-Befehl eingestellt.

4/6.5.4.1

QOKE — der 16-Bit-POKE

```
                    QOKE:
083A 20 42 04           JSR EGETWRD     ;ADRESSE HOLEN
083D 20 1E 04           JSR ECHKCOM     ;KOMMA PRUEFEN
0840 20 30 04           JSR EFRMEVL     ;WERT HOLEN
0843 20 DA 77           JSR CHKNUM      ;WERT HOLEN
0846 20 5D 04           JSR FLPNOS      ;-> INTEGER
0849 48                 PHA             ;MSB SICHERN
084A A9 16              LDA #POKER      ;ZIELVEKTOR
084C 8D B9 02           STA STAVEC      ;ERRICHTEN
084F 98                 TYA             ;LSB IN AKKU
0850 A0 00              LDY #0
0852 AE D5 03           LDX CURRENTB    ;BANKNUMMER
0855 20 77 FF           JSR INDSTA      ;LOW-BYTE SPEICHERN
0858 C8                 INY
0859 68                 PLA             ;HIGH-BYTE
085A AE D5 03           LDX CURRENTB    ;BANKNUMMER
085D 20 77 FF           JSR INDSTA      ;SPEICHERN
0860 60                 RTS             ;ENDE QOKE
                    ;
```

Listing 4/6.5.4-1

Die Zieladresse wird mit der Routine EGETWRD geholt, die in den Hilfszellen POKER, POKER+1 einen Vektor auf eben diese Zieladresse richtet. Den einzupokenden Wert könnte man zwar ebenfalls mit GETWRD holen, doch dann wäre unser schöner POKER-Vektor überschrieben. Deshalb lesen wir sie mit EFRMEVL und wandeln dann mit FLPNOS in eine ganze Zahl um, wobei diese im Akku und im Y-Register übergeben wird und somit POKER nicht zerstört wird.

Zum Speichern müssen wir die Routine INDSTA des Kernal benutzen, die das Speichern in eine beliebige Bank ermöglicht. Sie verlangt als Parameter in der Zelle STAVEC die Adresse des Vektors, im Y-Register den Offset der Speicherzelle (in unserem Fall 0 für das Lowbyte, 1 für as Highbyte), sowie im X-Register die Nummer der gewünschten Speicherbank, wobei hier wieder — wie beim BANK-Befehl — die Nummer der Konfiguration gemeint ist.

4/6.5.4.2

QEEK — der 16-Bit-PEEK

```
                QEEK:
                DHZ:        .EQ FACEXP+3     ;HILFSZEIGER
0861 20 27 04               JSR EPARCHK      ; ADRESSE IN () HOLEN
0864 20 DA 77               JSR CHKNUM       ; NUMERISCH PRUEFEN
0867 20 5D 04               JSR FLPNOS       ; -> INTEGER
086A 84 66                  STY DHZ          ; ZEIGER 'DHZ'
086C 85 67                  STA DHZ+1        ; ERRICHTEN
086E AE D5 03               LDX CURRENTB     ; BANKNUMMER
0871 A9 66                  LDA #DHZ         ; ZEIGERADRESSE
0873 A0 01                  LDY #1
0875 20 74 FF               JSR INDFET       ;HIGH-BYTE HOLEN
0878 48                     PHA              ;HIGH-BYTE SICHERN
0879 AE D5 03               LDX CURRENTB     ; BANKNUMMER
087C A9 66                  LDA #DHZ         ; ZEIGERADRESSE
087E 88                     DEY
087F 20 74 FF               JSR INDFET       ;LOW-BYTE HOLEN
0882 A8                     TAY              ;LSB -> Y
0883 68                     PLA              ;MSB -> A
0884 20 C9 84               JSR NOSFLT       ;ERGEBNIS -> FAC
0887 60                     RTS              ;ENDE QEEK
                ;
```

Listing 4/6.5.4-2

Diese Routine ist so etwas wie die Umkehrung des QOKE-Befehls aus dem vorigen Kapitel. Auch hier dürfen wir den POKE-0000-Vektor nicht überschreiben, da — wie bei allen Funktionen — Vorsicht geboten ist, weil ein rekursiver Aufruf nie auszuschließen ist. Als Vektor verwenden wir deshalb die Zellen des Fließkomma-Akkumulators, die in diesem Moment noch nicht gebraucht werden.

Das Laden der Speicherzellen geschieht mit der Kernal-Routine INDFET, die den Zeiger auf den Vektor mit der Ladeadresse im Akkumulator erwartet. X- und Y-Register werden wie im vorigen Kapitel vorbesetzt.

4/6.5.5

AUS-Erweiterungen abschalten

```
               ;** ERWEITERUNG AUSSCHALTEN **
               ;
               AUS:
0815 A9 A2             LDA #<NGONE
0817 8D 08 03          STA IGONE       ;IGONE-VEKTOR AUF
081A A9 4A             LDA #>NGONE     ;STANDARD-ROUTINE
081C 8D 09 03          STA IGONE+1     ;STELLEN
               ;
081F A9 DA             LDA #<NEVAL     ;ARITH.-VEKTOR AUF
0821 8D 0A 03          STA IEVAL       ;NORMALEN WERT
0824 A9 78             LDA #>NEVAL     ;STELLEN
0826 8D 0B 03          STA IEVAL+1
               ;
0829 78                SEI
082A A9 65             LDA #<NIRQ      ;IRQ-VEKTOR AUF
082C 8D 14 03          STA IIRQ        ;STANDARDWERT
082F A9 FA             LDA #>NIRQ      ;SETZEN
0831 8D 15 03          STA IIRQ+1
0834 58                CLI
               ;
0835 A9 04             LDA #$04
0837 85 2F             STA VARTAB      ;VARIABLEN AB $10400
0839 60                RTS             ;FERTIG
               ;
               ;
```

Listing 4/6.5.5-1

Dieser Befehl macht nichts anderes als die Vektoren, die bei der Initialisierungs-Routine verbogen wurden, nun wieder auf die Standard-Routine zu richten. Im einzelnen handelt es sich um den Befehlsausführ-Vektor, den Arithmetik-Vektor, den IRQ-Vektor und den Zeiger auf den Beginn der Variablen in der Bank 1.

4/6.5.6

Zusammenfassung der Basic-Erweiterungen zu ladbaren Dateien

Die in den vorangegangenen Kapiteln erläuterten Basic-Erweiterungen, der in Kapitel 6.3.1 vorgestellte Interruptmanager und die in Kapitel 6.3.2 angeführten Zahlkonvertierungen belegten im Speicher die in nachfolgender Tabelle aufgeführten Bereiche:

Dekoder für Befehle und Funktionen:	$1300 bis $14F1 in Bank 0
Interruptmanager:	$1303 bis $1348 in Bank 0
Zahlkonvertierungen und andere Hilfsroutinen:	$0400 bis $05C0 in Bank 1
Neue Befehle und Funktionen:	$0800 bis $09FD in Bank 1

Diese Programme lassen sich in insgesamt zwei ladbare Maschinenprogrammdateien zusammenfassen, eine für Bank 0 und eine für Bank 1. In der Belegung der Bank 1 klafft eine Lücke zwischen $05C0 und $0800. Dieser Platz ist vorgesehen, um weitere Hilfsroutinen aufzunehmen.

Den verbleibenden Zwischenraum muß man im vorliegenden Fall auffüllen, damit eine zusammenhängende Datei entsteht. Dies geht mit dem in diesem Buch beschriebenen Assembler am einfachsten, indem man folgende zwei Zeilen vor die Sprungtabelle der Befehle setzt, die bei $0800 beginnt.

```
434    MARK1:     .EQ *
435               .DB $0800-MARK1
```

Listing 4/6.5.6-1

Verwendet man den in diesem Buch beschriebenen Assembler, so erhält man die Maschinenprogrammdateien durch entsprechende .ou-Befehle. Nennt man diese Dateien z.B. „erw.Bank0" und „erw.Bank1", so kann man sich auch noch ein kleines Startprogramm in Basic schreiben, das die Erweiterungen lädt und startet. Dies könnte dann wie folgt aussehen:

```
10 BLOAD"ERW.BANK0" ON B0
20 BLOAD"ERW.BANK1" ON B1
30 SYS4864
40 NEW
```

Listing 4/6.5.6-2

Um Ihnen für zukünftige Erweiterungen eine Hilfestellung zu geben, bilden wir im folgenden die Tabelle der Symbole ab, die bei der Assemblierung der Basic-Erweiterungen entstanden sind. Dabei wurden die Programme wie oben vorgeschlagen aufgeteilt, so daß insgesamt zwei verschiedene Symboltabellen ausgedruckt sind, eine für die Programme in Bank 0 und die andere für Programme in Bank 1.

Symbole der Programme in Bank 0:

```
ACPTR      U FFA5  ARYTAB     U 0031  AUS        P 0815  AUSW2      P 094E
AUSW21     P 0974  BINC1      P 0519  BINCONV    P 0514  BININT     P 04AA
BININT1    P 04B2  BININT2    P 04C3  BININT9    P 04CA  BINSTR     P 057B
BINVAL     P 0572  BZPLY      P 04E7  BZPLY1     P 04F1  CGETL      U 9109
CHKCLS     U 7956  CHKCOM     F 795C  CHKIN      U FFC6  CHKNUM     F 77DA
CHKOPN     U 7959  CHKOUT     U FFC9  CHRGET     U 0380  CHRGOT     U 0386
CHRIN      U FFCF  CHROUT     U FFD2  CIA2       F DD00  CIOUT      U FFA8
CLOCKNR    P 0808  CLOSE      U FFC3  CLRCHN     U FFCC  COIN       U 90FD
COMBYT     U 8809  COOUT      U 90E8  CURRENTB   F 03D5  DHZ        F 0066
ECHKCOM    P 041E  ECHRGET    P 0454  EFRMEVL    P 0430  EGETBYT    P 0439
EGETNUM    U 0448  EGETWRD    P 0442  EPARCHK    P 0427  ERROR      U 4D3C
FACEXP     F 0063  FACSGN     F 0068  FCERR      F 7D28  FETVEC     U 02AA
FK100      P 0888  FLOATS     U 8C70  FLPINT     U 849F  FLPNOS     P 045D
FMULT      F 8A24  FRESTR     F 877E  FRETOP     U 0035  FRMEVL     F 77EF
FRMNUM     U 77D7  FRMSTR     U 877B  FZ1        F 00FA  GETADR     U 8815
GETBYT     F 87F4  GETCFG     U FFCB  GETIN      U FFE4  GETNUM     F 8803
GETSPA     F 9299  GETWRD     F 8812  GIVAYF     U 793C  GTBYTC     U 87F1
H1         F 0063  H2         F 0064  H3         F 0065  HEXC1      P 04F7
HEXCONV    P 04F2  HEXINT     P 0472  HEXINT1    P 047A  HEXINT2    P 0492
HEXINT9    P 04A9  HEXVAL     P 0569  HILF       P 08DA  ICRNCH     U 0304
IERROR     U 0300  IESCEX     U 0310  IESCLK     U 030C  IESCPR     U 030E
IEVAL      F 030A  IGONE      F 0308  IIRQ       F 0314  ILLQ       P 046F
IMAIN      U 0302  INCHR      U 90E5  INCTXT     P 04E0  INCTXT1    P 04E6
INDCMP     U FF7A  INDEX      F 0024  INDFET     F FF74  INDIN1R1   F 03B7
INDSTA     F FF77  INDTXT     U 03C9  INTBIN     P 04CB  INTBIN1    P 04D1
INTFLP     P 0562  INTHEX     U 0598  INTHEX1    P 059E  INTHEX2    P 05A8
IQPLOP     U 0306  LEN1       F 866E  LINPRT     U 8E32  LISTEN     U FFB1
LOAD       U FFD5  MARK1      F 05C1  MAXMEM1    U 0039  MMUCR      F FF00
MOVMF      U 8C00  NEVAL      F 78DA  NEWSTT     U 4AF6  NGONE      F 4AA2
NIRQ       F FA65  NOSFLT     F 84C9  OPEN       U FFC0  OUTCH      U 90DF
PARCHK     F 7950  PAUEND     P 08D7  PAULOOP    P 08CB  PAUSE      P 088D
PLSV       U 91AE  POKER      F 0016  PTRGET     U B08B  PUFFER     F 0100
PUTNEW     F 86E3  QEEK       P 0861  QINT       F 8CC7  QOKE       P 083A
RA1ROMIO   F 0040  READST     U FFB7  REALSP     U 5604  RUECKH     F 0003
RUECKL     F 00B2  SAVE       U FFD8  SECOND     U FF93  SETBNK     U FF68
SETLFS     U FFBA  SETT2      P 092A  SETTIM1    P 08F2  SETTIME    P 08DB
SETTPM     U 0928  SNERR      U 796C  SNGFLT     U 8404  STAVEC     F 02B9
STOP       F FFE1  STREND     U 0033  STRLIT     F 869A  STROUT     U 55E2
SWRAM0     U FF01  SWRAM1     F FF02  SWROMRA0   U FF03  SWROMRA1   F FF04
TALK       U FFB4  TEST       P 09FD  TIME       P 0981  TIME3      P 09AF
TIME4      P 0965  TIMEU1     P 09D4  TIMEU2     P 09D8  TIMEU3     P 09DC
TIMEUD     P 09EF  TKSA       U FF96  TXTPTR     F 003D  TXTTAB     U 002D
UNLSN      U FFAE  UNTALK     U FFAB  USRPOK     U 1218  VAL        F 804A
VALNEU     P 0536  VALPAR     P 0553  VALPAR1    P 0555  VALTYP     F 000F
VARPNT     U 0047  VARTAB     F 002F  ZAEHL      P 08D9
```

Listing 4/6.5.6-3

Symbole der Programme in Bank 1:

```
ACPTR      U FFA5  AKKUSICH  P 136E  ARYTAB     U 0031  AUS        F 0800
BEBAS      P 1396  BEBASC    P 1395  BEFNR      P 136F  BINCON     P 148E
BINCONV    F 0403  BINSTR    F 0406  BTAB       P 13E1  CGETL      U 9109
CHKCLS     U 7956  CHKCOM    U 795C  CHKIN      U FFC6  CHKNUM     U 77DA
CHKOPN     U 7959  CHKOUT    U FFC9  CHRGET     F 0380  CHRGOT     U 0386
CHRIN      U FFCF  CHROUT    U FFD2  CIOUT      U FFA8  CLOSE      U FFC3
CLRCHN     U FFCC  COIN      U 90FD  COMBYT     U 8809  COOUT      U 90EB
CURRENTB   U 03D5  DECBZ     P 1365  DECBZ1     P 136B  EEVAL      P 13FF
EG1        P 1388  EG2       P 139F  EG3        P 13B4  EG4        P 13CD
EGONE      P 1370  EIRQ      P 131B  ERROR      U 4D3C  FACEXP     U 0063
FACSGN     U 0068  FCERR     U 7D28  FE1        P 1427  FE2        P 143E
FE3        P 1453  FE4       P 146A  FETVEC     U 02AA  FKTVAL     P 1474
FLOATS     U 8C70  FLPINT    U 849F  FMULT      U 8A24  FNBAS      P 1435
FNBASC     P 1434  FNNR      P 13FE  FRESTR     U 877E  FRETOP     U 0035
FRMEVL     U 77EF  FRMNUM    U 77D7  FRMSTR     U 877B  FSTAB      P 149B
FTAB       P 14A3  GETADR    U 8815  GETBYT     U 87F4  GETCFG     U FFCB
GETIN      U FFE4  GETNUM    U 8803  GETSPA     U 9299  GETWRD     U 8812
GIVAYF     U 793C  GTBYTC    U 87F1  HEVAL      F 03FA  HEVALCODE  U 1353
HEXCON     P 1481  HEXCONV   F 0400  IADR       P 1317  ICRNCH     U 0304
IDUMMY     P 131A  IERROR    U 0300  IESCEX     U 0310  IESCLK     U 030C
IESCPR     U 030E  IEVAL     F 030A  IGONE      F 0308  IIRQ       F 0314
IMAIN      U 0302  INCHR     U 90E5  INDCMP     U FF7A  INDEX      U 0024
INDFET     U FF74  INDIN1R1  U 03B7  INDSTA     U FF77  INDTXT     U 03C9
INIT       P 14B8  INTAUSF   P 1333  INTAUSF1   P 1335  INTMAN1    P 132B
INTMANS    P 1320  IOPLOP    U 0306  IRQM       P 1306  IRQZ       P 1319
ISPTAB     P 1307  LESP      U 0B03  LEST       U 0B00  LINPRT     U 8E32
LISTEN     U FFB1  LOAD      U FFD5  MAXMEM1    U 0039  MELDUNG    P 14E7
MMUCR      U FF00  MOVMF     U 8C00  NEVAL      F 78DA  NEWSTT     F 4AF6
NGONE      F 4AA2  NIRQ      F FA65  NOSFLT     U 84C9  OPEN       U FFC0
OUTCH      U 90DF  PARCHK    U 7950  PAUSE      F 0806  PLSV       U 91AE
POKER      U 0016  PTRGET    U B08B  PUTNEW     U 86E3  QEEK       F 0809
QINT       U 8CC7  QOKE      F 0803  READST     U FFB7  REALSP     U 5604
RUECKH     U 0003  RUECKL    U 00B2  SAVE       U FFD8  SECOND     U FF93
SETBNK     U FF68  SETLFS    U FFBA  SETTIME    F 080C  SNERR      U 796C
SNGFLT     U 84D4  STAB      P 13D7  STAVEC     U 02B9  STOP       U FFE1
STREND     U 0033  STRLIT    U 869A  STROUT     F 55E2  SWRAM0     F FF01
SWRAM1     U FF02  SWROMRA0  F FF03  SWROMRA1   F FF04  TALK       U FFB4
TEST       F 0812  TIME      F 080F  TKSA       U FF96  TXTPTR     F 003D
TXTTAB     U 002D  UNLSN     U FFAE  UNTALK     U FFAB  USRPOK     U 1218
VALNEU     F 0409  VALTYP    F 000F  VARPNT     U 0047  VARTAB     F 002F
VIEW       U 0806  WCC1      P 135B  WCL        F 0010  WCO        F 0004
WCODECOP   P 1359  WCZ       P 134D  WEITCODE   P 1349  WEITSPR    F 03F0
ZIEL       F 03F4
```

Listing 4/6.5.6-4

4/7
Tips & Tricks

In Kapitel 4/7 wollen wir in loser Folge Tips und Tricks — getrennt für C 64 und C 128 aufführen, die Ihnen entweder das Programmieren oder die Fehlersuche vereinfachen oder beim Vermeiden von Fehlern helfen oder sie über im Handbuch nicht Erläutertes informieren oder . . .

4/7.1
Tips & Tricks zum C 64

Wir beginnen mit zwei Themen, die von allgemeiner Bedeutung sind: Speicherplatz- und Rechenzeiteinsparung bei Basic-Programmen.

4/7.1.1
Speicherplatz sparen

Das Einsparen von Rechenzeit und Speicherplatz sind meist konkurrierende Ziele, in manchen Fällen kann man aber auch durch Sparen von Speicherplatz die Rechenzeit reduzieren, wie zum Beispiel beim Weglassen von Bemerkungen.

Obwohl heute auch Home-Computer über mehr Speicher verfügen als noch vor einigen Jahren, nutzen die Anwender sehr oft den verfügbaren Speicher bis ins Letzte aus. Dies wird zu einem großen Teil durch die Anwenderfreundlichkeit der Programme notwendig, da hier in der Regel Erklärungstexte und Grafiken den Speicherplatz verbrauchen. Da der C 64 nach heutigen Maßstäben sowieso nicht verschwenderisch mit Speicherplatz ausgestattet ist, ist die Beachtung der folgenden Tips in manchen Fällen unausweichlich.

Bevor wir zu den Tips kommen, noch ein paar Worte zur Thematik im allgemeinen: Nicht immer ist es sinnvoll, Speicherplatz zu sparen, weil dadurch meist die Möglichkeit der Eigendokumentation von Programmen sehr stark eingeschränkt, wenn nicht sogar unmöglich gemacht wird. Wenn man den in diesem Werk vorgestellten REM-KILL verwendet, kann man in der Programmierphase ein reichlich dokumentiertes Programm erstellen, und dies mit dem REM-KILL von Speicherfressern wie REM-Zeilen befreien.

Man darf aber — wie eingangs erwähnt — nicht nur das Ziel der Speicherplatzoptimierung verfolgen, sondern dabei die Rechenzeit nicht aus den Augen lassen, so daß ein Mittelweg gefunden werden muß.

Abgesehen von den logischen Möglichkeiten der Speicherplatzeinsparung (vgl. Datum platzsparend speichern), die immer nur anhand der Problemstellung abzuschätzen sind, sollen im folgenden einige allgemeine Tips gegeben werden:

Schreiben Sie mehrere Anweisungen in eine Zeile

Jede Zeile benötigt 5 Bytes zur internen Verwaltung:

— 2 Byte für die Zeilennummer
— 2 Byte für den Verweis auf die nächste Zeile im RAM
— 1 Byte für die Zeilenendmarkierung

Durch jede eingesparte Zeile gewinnen Sie 4 Byte.

Beispiel:

```
100 FOR I = 1 TO N
110     PRINT N
120 NEXT

oder

100 FOR I = 1 TO N : PRINT N : NEXT
```

sparen Sie 8 Byte: 2*5 für die Zeile abzüglich 2*1 für die ':'-Zeichen. Allerdings ist die kürzere Möglichkeit vom Standpunkt der Dokumentation die schlechtere, wegen des fehlenden Überblickes.

Löschen Sie alle unnötigen Leerzeichen

Jedes Leerzeichen — außer dem Leerzeichen zwischen Zeilennummer und der ersten Anweisung — muß intern gespeichert werden. Mit:

```
100 FORI=1TON:PRINTN:NEXT
```

sparen Sie also gegenüber dem letzten Beispiel weitere 9 Byte. Dies erledigt übrigens der schon erwähnte REM-KILL auch.

Man sieht schon an diesem kleinen Beispiel, daß durch Auslassen von Leerzeichen ein relativ großer Prozentsatz Speicher eingespart werden kann.

Löschen Sie alle REM-Befehle

Jeder REM-Befehl benötigt ein Byte für das Basic-Token zuzüglich der Anzahl der Zeichen hinter dem REM. Steht das REM in einer gesonderten Zeile, sogar noch weitere 5 Byte mehr (s. o.).

Benutzen Sie Variablen statt Konstanten

Wenn Sie in einem Programm 10 mal den Wert 1.23456789 benötigen, setzen Sie einmal A=123456789 und benutzen Sie die Variable A statt der Konstanten 1.23456789.

Jedes Zeichen einer Konstanten benötigt ein Byte Speicherplatz. Auch Zahlkonstanten werden zeichenweise gespeichert.

Benutzen Sie temporäre Variable häufiger

Kurzlebige Variablen (z.B. sofortiges Verarbeiten nach dem Einlesen wie beim Einlesen eines einzelnen Zeichens mit A$) sollten häufiger in verschiedenen Programmteilen verwendet werden, da BASIC keine lokalen Variablen, wie sie zum Beispiel bei blockorientierten Sprachen (PASCAL) vorkommen, kennt, und die sogenannte Lebensdauer im Speicher für eine Variable von der Definition bis zum Programmende fortläuft. Durch die Wiederverwendung von Variablen sparen Sie einerseits Platz in der Variablenliste, andererseits Rechenzeit, da das Durchsuchen der rechnerinternen Liste der Variablen schneller abläuft.

Häufig wiederkehrende Programmteile als Unterprogramm schreiben

Ein GOSUB XXXXX mit zugehörigem RETURN braucht viel weniger Speicherplatz als die mehrmalige Programmierung desselben Programmstückes.

Setzen Sie häufig benutzte Unterprogramme an den Anfang

Diese Unterprogramme können durch ein GOTO „erste Zeile nach den Unterprogrammen" sehr einfach übersprungen werden, aber das Voranstellen hat einen Vorteil: Die Zeilenangabe nach einem GOSUB wird zeichenweise gespeichert, d.h. GOSUB 3 benötigt 4 Byte weniger Speicher als GOSUB 30 000. Wird ein Unterprogramm sehr oft aufgerufen, macht sich diese Ersparnis jedesmal bemerkbar.

Dies spart außerdem auch noch Rechenzeit, da bei einem Unterprogrammaufruf ab dem Anfang des Programmes nach dessen Startzeile gesucht wird, wenn die gesuchte Zeile kleiner als die aktuell bearbeitete Zeile ist. Andernfalls wird ab der aktuellen Zeile gesucht.

Wird das Unterprogramm von verschiedenen Stellen im Hauptprogramm aufgerufen, ist es günstiger, dies an den Programmanfang zu setzen. Dies trifft auch zu, wenn z.B. das Unterprogramm mit vielen GOTO und GOSUB versehen ist, die auf frühere Zeilennummern zeigen. Die Einsparung liegt bei großen Programmen dadurch nicht selten im Sekunden-Bereich.

Benutzen Sie die 0-Elemente von Feldern

In BASIC kann zwar die Feldlänge mit DIM A(x) bestimmt werden, wobei x eine positive ganze Zahl darstellt, jedoch wird damit nur das letzte Element festgelegt. Die Felder beginnen immer mit dem Element „0". Statt DIM A(100) — wenn 100 Elemente gewünscht sind — genügt also DIM A(99).

Wie leicht zu sehen ist, leidet durch die Speicherplatzeinsparung die Übersichtlichkeit der Programme sehr. Bevor Sie also zu obigen Hilfsmiteln greifen, sollten Sie andere Möglichkeiten (z.B. Modularisierung) in Betracht ziehen.

4/7.1.2

Rechenzeit verkürzen

Die Rechenzeit läßt sich am besten durch eine genaue Problemanalyse und entsprechende Programmierung verkürzen. Aber auch allgemeine Regeln, wie das Löschen von REM-Zeilen bzw. -befehlen bringen eine Rechenzeitersparnis.

Optimierung geschachtelter Schleifen

Besonders wichtig bei der Verkürzung von Rechenzeit sind geschachtelte Schleifen. Zum einen bringen kleinere Optimierungen bei den innersten Schleifenanweisungen sehr viel, zum anderen kann man durch genaues Überlegen bei den Schleifengrenzen einige Zeit sparen. Dazu je ein Beispiel:

Meist genügt es, in geschachtelten Programmschleifen die Anweisung innerhalb der innersten Schleife zu optimieren:

```
100 FOR I = 1 TO 10
110     FOR J = 1 TO 10
120         FOR K = 1 TO 10
130             FOR L = 1 TO 10
140                 FOR M = 1 TO 10
150                        Anweisungen
160                 NEXT
170             NEXT
180         NEXT
190     NEXT
200 NEXT
```

Obwohl die einzelnen Schleifen es nur auf je 10 Durchläufe bringen, macht sich jede im Teil „Anweisungen“ eingesparte Millisekunde in der Rechenzeit mit 100 Sekunden bemerkbar, da der Anweisungsteil 100.000mal durchlaufen wird.

Beachten Sie auch, daß die Schleifen-Grenzen entsprechend der Problemstellung gewählt werden. Beim obigen Beispiel etwa wäre zu überlegen, ob z.B. die Laufvariable J nicht nur von 2 bis 8 laufen muß. Bei einer „Anweisungen"-Rechenzeit von 10 Sekunden werden dadurch insgesamt 3000 Sekunden = 50 Minuten eingespart.

Durch ungeschickte Programmierung in diesen Punkten sind „normale" Rechenzeiten von Stunden sehr leicht auf Jahre zu verlängern und werden damit undurchführbar. Ein konkretes Beispiel: Sie wollen „irgendetwas" bei den Lottozahlen ausprobieren, und müssen dazu alle Zahlenkombinationen durchlaufen. Eine mögliche Vorgehensweise:

```
100 FOR I=1 TO 49
110     FOR J=1 TO 49
120         FOR K=1 TO 49
130             FOR L=1 TO 49
140                 FOR M=1 TO 49
150                     FOR N=1 TO 49
160                         GOSUB <prüfen>
                            <prüfen auf zulässige Kombination,
                             d.h. alle Zahlen paarweise ver-
                             schieden>
                            <wenn unzulässig, Schleife verlassen>
                            <Permutationen (gleiche Zahlen, nur in
                             anderer Reihenfolge) überspringen>
170                         GOSUB <bearbeiten>
                            <Anweisungen>
180                     NEXT
190                 NEXT
200             NEXT
210         NEXT
220     NEXT
230 NEXT
```

Die Anweisungen — und insbesondere das Prüfen — können vielleicht eine Sekunde den Rechner in Anspruch nehmen. Dabei wird dieser Teil 49*49*49*49*49*49 durchlaufen, was ca. 13.841.287.200 Sekunden oder etwa 439 Jahren entspricht. So ein Programm lohnt schon gar nicht den Programmieraufwand.

Wenn Sie die inneren Anweisungen vielleicht so optimieren können, daß nur noch 1/10 Sekunde benötigt wird, sieht die Sache zwar besser aus, ist aber immer noch undurchführbar.

Man braucht aber nur die Schleifenkriterien etwas zu durchdenken, und stellt fest, daß mit folgender Konstruktion immer noch alle möglichen Konstellationen — und einige zuviel — erreicht werden:

```
100 FOR I=1 TO 44
110     FOR J=2 TO 45
120         FOR K=3 TO 46
130             FOR L=4 TO 47
140                 FOR M=5 TO 48
150                     FOR N=6 TO 49
160                         GOSUB <prüfen, wie oben>
170                         GOSUB <bearbeiten>
180                     NEXT
190                 NEXT
200             NEXT
210         NEXT
220     NEXT
230 NEXT
```

Es sind jetzt nur noch 44^6 Bearbeitungen der inneren Schleife durchzuführen. Bei einer angenommenen Bearbeitungszeit von 1/10 Sekunde wären dies circa 72.563.138 Sekunden, entsprechend 2 Jahre und etwa 4 Monate. Jetzt könnte man es vielleicht mit einem guten, absturzsicheren Computer schaffen, wenn man sonst nichts mehr damit machen will!

Man kann das Ganze aber nochmals überdenken, und erhält vielleicht das Ergebnis auf der folgenden Seite.

```
100 FORI=1TO44
110 FORJ=I+1TO45
120 FORK=J+1TO46
130 FORL=K+1TO47
140 FORM=L+1TO48
150 FORN=M+1TO49
160 GOSUB <bearbeiten>
170 NEXT
180 NEXT
190 NEXT
200 NEXT
210 NEXT
220 NEXT
```

Das Ergebnis ist immer noch eine vollständige Liste der möglichen Lottokombinationen. Die Anzahl der Berechnungen ist nun allerdings etwas schwerer zu ermitteln: Die erste Schleife (I) wird 44mal durchlaufen. Die nächste Schleife zuerst 44mal (von 2 bis 45), dann 43mal, dann 42mal, ..., bis zum einmaligen letzten Durchlauf (I=44, K=45). Nach Gauß (n*(n+1)/2, n ist bei uns 44) ergibt dies 990 Durchläufe. Dieser Wert wird allerdings nicht mit 44 multipliziert, sondern würde allein bei den beiden Schleifen (I, J) schon die Gesamtzahl der inneren Läufe darstellen.

Wenn man diese Rechenweise fortführt, oder die Kombinatorik zu Hilfe nimmt, kommt man auf 13.983.816 Durchläufe, was exakt mit der Anzahl der möglichen Kombinationen übereinstimmt, wie man leicht nachvollziehen kann, wenn man die Werte der Laufvariablen im Inneren der Schleife in Gedanken auf einen Lottoschein überträgt: Erst alle Zahlen von 1 bis 6, dann wird statt 6 die 7 angekreuzt darauf die 8, bis hin zur 49. Anschließend wird statt der bisherigen 5 die 6 markiert und als sechste Zahl wieder 7, die wieder durch 8 ... 49 ersetzt wird.

Man hat aber jetzt nicht nur Durchläufe eingespart, sondern das Unterprogramm zum Prüfen kann ebenso entfallen, da keine Permutationen (Zahlenverdrehungen) und doppelte Zahlen mehr auftreten.

Wenn wir trotzdem eine Rechenzeit von 1/10 Sekunde pro inneren Durchlauf ansetzen, so wären wir — ohne Stromausfall und andere Katastrophen — in gut 16 Tagen fertig!

Kommen wir an dieser Stelle nochmals auf die Problemanalyse zurück. Zur Darstellung des Sachverhaltes bei einer Schleifenschachtelung haben wir natürlich ein extremes Beispiel herangezogen. Auch 16-17 Tage sind eine lange Rechenzeit,

und man kann nie sicher sein, ob die Frau des Hauses nicht gerade *die* Steckdose für den Staubsauger braucht.

Man muß schließlich nicht für einen Systemtest *alle* Kombinationen durchprobieren. Man kann sich auch zufällige Zahlen ziehen, und es bei 1.000, 10.000 oder auch 100.000 Ziehungen bewenden lassen. Eine solche Stichprobe reicht mit an Sicherheit grenzender Wahrscheinlichkeit aus. In der — ernsthaften — Spieltheorie nennt sich ein solcher Test eine Monte-Carlo-Simulation.

Sollten Sie im konkreten Fall dennoch die erstgenannte Methode anwenden wollen, vermeiden Sie PRINT-Befehle innerhalb der innersten Schleifen, auch diejenigen, die zur Kontrolle dienen, ob der Rechner noch läuft. In solchen Fällen sind PRINT-Befehle ein entscheidender Zeitfresser. Bei einer einzigen Anweisung in der innersten Schleife (O=O+1) und PRINT-Befehle für das aktuelle Zwischenergebnis jeweils in den drei äußeren Schleifen, benötigt der C 64 (C 128 im 64er-Modus) schon 23 Stunden (ohne Abschalten VIC siehe unten). Der C 128 benötigt für diese Berechnung im SLOW-Modus mehr als 31 Stunden und im FAST-Modus ungefähr 13 Stunden.

Vermeiden von unnötigen Leerzeichen und REM-Befehlen

Diese beiden Möglichkeiten gehen ausnahmsweise konform mit der Speicherplatzeinsparung. Im normalen Programm, das nur einmal oder wenige Male durchlaufen wird, bringt das wenig, aber eine Millisekunde Ersparnis im letzten Beispiel schafft eine kürzere Wartezeit von ungefähr 4 Stunden. Und dies allein dadurch, daß der Rechner (in Basic der Interpreter) keine unnötigen Zeichen zu überlesen braucht.

Bei Kompilern, die beim Kompilieren sowieso diesen „Ballast" nicht bearbeiten, spielen Leerzeichen und Kommentare keine Rolle und sollten zur Dokumentation verwendet werden.

Variablen statt Konstanten

Das Einsetzen von Variablen statt Konstanten spart ebenfalls Rechenzeit, da eine Konvertierung der Daten entfällt (etwa Faktor 10). Häufig benutzte Variablen sollten am Programmanfang definiert werden (z.B. einfach mit A=0), da sie dann beim Suchen in der Variablentabelle schneller gefunden werden.

NEXT ohne Index-Variable

Benutzen sie NEXT ohne Index-Variable (also nicht NEXT I), da somit die Überprüfung durch das Betriebssystem entfällt.

VIC abschalten

Haben Sie auch schon festgestellt, daß der Bildschirm nichts anzeigt, wenn ein Programm von Kassette geladen wird? Da der Rechner den VIC nicht benötigt, wird er einfach abgeschaltet. Weil Eingaben von der Tastatur damit auch sinnlos sind und Timer etc. nicht gebraucht werden, kann man auch den IRQ abschalten. Zusammen ergibt dies:

```
POKE 53265, PEEK(53265) AND 239
        (VIC ausschalten)
POKE 56333,1
        (IRQ abschalten)
```

und das Ganze zurück:

```
POKE 53265, PEEK(53265) OR 16
POKE 56333,129
```

Ein kleines Problem: Die Tastatur geht nicht mehr.

4/7.1.3

ASCII-Code in Bildschirmcode umwandeln

Mit den ersten beiden Zeilen des folgenden Programmes wird ein in A vorliegender ASCII-Code in den Bildschirmcode des Commodore 64 umgewandelt, der Rest stellt ein Testprogramm dar:

```
 4 A(0)=128 : A(1)=32 : A(2)=0 : A(3)=64 : A(4)=192 :
              A(5)=96 : A(6)=64 : A(7)=96
 8 DEF FNB(X)=A(X/32)+(XAND31)+33*(X=255)
10 FOR X=25 TO 255
15 PRINT"(CLR)",X,FNB(X),CHR$(34),CHR$(X)
20 POKE 1048,FNB(X)
25 FOR W=1 TO 200 : NEXT
30 NEXT
```

In der ersten Zeile wird eine Umrechnungsvorgabe in einem Feld festgelegt, die in der zweiten Zeile verwertet wird. Bei der Umrechnung können jeweils Bereiche von 32 Zeichen als ganzes behandelt werden (X AND 31), wenn man von dem letzten Zeichen (PI: ASCII=255 / Bildschirmcode=94) absieht. Näheres ist aus folgender Tabelle ersichtlich:

Bits im ASCII-Code			**Bit im Bildschirmcode**		
128	64	32	128	64	32
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	1	1
1	1	0	0	1	0
1	1	1	0	1	1

Man sieht leicht, daß die Zuordnung nicht umkehrbar eindeutig ist.

4/7.1.4

Bildschirm-RAM und Zeichensatz verlegen

Das Register 53272 ist für die Position des Bildschirm-RAM und den Zeichensatz zuständig. Dabei gilt:

1. Halbbyte (Bits 7-4)*1024 = Startadresse Bildschirm-RAM
2. Halbbyte (Bits 3-0)*1024 = Startadresse Zeichengenerator

Achtung: Bit 0 wird bei der Startadresse des Zeichengenerators nicht berücksichtigt, so daß sich folgende Möglichkeiten ergeben:

Wert in 53272	Startadresse
xxxx000x	0
xxxx001x	2048
xxxx010x	4096
xxxx011x	6144
xxxx100x	8192
xxxx101x	10240
xxxx110x	12288
xxxx111x	14336

Beispiel

```
POKE 53272, 18 (=1*16+2)
```

ergibt Startadresse Bildschirm-RAM : 1024
Startadresse Zeichensatz : 2048

Achtung: für das Bildschirm-RAM muß auch die Adresse 648 geändert werden:

```
POKE 648, (Startadresse)/256
```

Das ist die relative Adresse zum VIC-Arbeitsbereich, der in 56576 definiert ist.

4/7.1.5

Schließen aller offenen Dateien
Anzahl der offenen Dateien

Mit

```
SYS 65511
```

kann man alle offenen Dateien schließen, allerdings nur im Rechner. Bei der Floppy geschieht dies mit

```
OPEN1,8,15 : CLOSE1
```

Die Anzahl der offenen Dateien läßt sich mit

```
PEEK(152)
```

feststellen.

4/7.1.6

Zeilen automatisch löschen — DELETE

Wer nach einem teilweise Umarbeiten eines Programmes viele Zeilen zuviel hat, wird sich sehnlichst eine DELETE-Routine wünschen. Im folgenden stellen wir eine Basic-Routine etwas ausführlicher vor, weil sie einige Tricks beinhaltet. Sicherlich geht es auch kürzer.

Verwendet wird der Tastaturpuffer und der Puffer für den Kassettenrekorder. Prinzip des Programmes:

- Zeilennummern, die gelöscht werden sollen, nacheinander auf den Bildschirm bringen
- Programm verlassen
- Zeile löschen
- Programm „mittendrin" wieder neu starten, natürlich mit der nächsten zu löschenden Zeile

Kurz gesagt: Der Handbetrieb wird automatisiert.

```
    1 GOTO 60100
60000 INPUT "VON";VO
60010 INPUT "BIS";BI
60020 INPUT "SCHRITTWEITE";SR
60030 POKE 831,VO AND 255
60040 POKE 832,VO/256
60050 POKE 833,BI AND 255
60060 POKE 834,BI/256
60070 POKE 835,SR
60100 VO = PEEK(832)*256 + PEEK(831)
60110 BI = PEEK(834)*256 + PEEK(833)
60120 SR = PEEK(835)
60130 PRINT"(CLR)";VO
60140 IF VO => BI THEN STOP
```

```
60150 VO = VO+SR
60160 POKE 831,VO AND 255
60170 POKE 832,VO/256
60180 POKE 631,145     : REM CURSOR NACH OBEN
60190 POKE 632,145     : REM CURSOR NACH OBEN
60200 POKE 633,145     : REM CURSOR NACH OBEN
60210 POKE 634,13      : REM CHR$(13)
60220 POKE 635,82      : REM R
60230 POKE 636,85      : REM U
60240 POKE 637,78      : REM N
60250 POKE 638,13      : REM CHR$(13)
60260 POKE 198,8       : REM ZAHL DER ZEICHEN IM PUFFER
```

Zeile 1 wird benötigt, weil der Tastaturpuffer nur zehn Zeichen aufnehmen kann und eine Zeilennummer für ein Programm ziffernweise (genauer: zeichenweise) eingegeben werden muß. Die Zeilen von 60000 bis 60070 werden nur zur Erfassung der gewünschten Daten benötigt, wobei die größeren Zahlen (VO, BI) jeweils in der üblichen Form (Low-Byte, High-Byte) in den Kassettenpuffer geschrieben werden.

In Zeile 60100 beginnt die eigentliche „Programmschleife“: Lesen der aktuell zu löschenden Zeilennummer, Endekriterium prüfen, neuen Wert anhand der Schrittweite berechnen, Sichern dieses Wertes, Füllen des Tastaturpuffers wie angegeben und Programmabbruch.

Warum die Variableninhalte sichern? Das Löschen einer Zeile ist eine Programmänderung und löscht somit alle Variablen. Wenn Sie die Zeile 1 weglassen, können Sie die Programmsequenz mit dem weiter unten beschriebenen KurzMERGE einladen, Zeile 1 eintippen und mit RUN 60000 starten.

Vielleicht gibt Ihnen das Programm auch einen Tip für Ihre anderen Probleme.

4/7.1.7

INPUT-Ersatz

Für Textverarbeitungsprogramme und Datenverwaltungsroutinen ist der normale Basic-Input-Befehl schlecht geeignet, da Probleme auftauchen, wenn Kommata, Doppelpunkte und Anführungszeichen im Text gemischt auftreten können.

Um diesen Mißstand abzuhelfen, können Sie für Eingaben am Bildschirm folgende Routine verwenden:

```
1000 B$ = ""              : REM Ausgabestring annullieren
1010 SYS 65487            : REM KERNAL-Routine CHRIN ($FFCF)
1020 A=PEEK(780)          : REM Akku = gelesenes Zeichen
1030 IF A=13 THEN RETURN  : REM Ende bei 'Return'-Taste
1040 B$=B$+CHR$(A)        : REM in B$ Zeichen sammeln
1050 GOTO 1010            : REM Schleife
```

Diese Routine wird mit GOSUB 1000 aufgerufen und gibt den eingelesenen String in B$ zurück. Hierbei wird kein Fragezeichen — wie bei INPUT üblich — ausgegeben, außerdem steht der Cursor nachher in der alten Zeile. Wenn es notwendig sein sollte, können Sie diesen Mangel aber leicht mit PRINT-Befehlen beseitigen.

Die CHRIN-Routine bewirkt beim ersten Aufruf, daß der Cursor blinkt und die Zeile eingegeben werden kann. Anschließend wird die Routine aufgerufen, um die Zeichen zu holen, wobei das Endekriterium das Auftauchen von CHR$(13) (RETURN-Taste) ist.

4/7.1.8

Joystick-Simulation über Tastatur

Da Joystick- und Tastaturabfrage gleichzeitig vorgenommen werden können, ist es auch möglich, die Joystickeingabe über Tastatur zu simulieren, was allerdings für schnelle Reaktionen einiges Geschick erfordert.

Bei untenstehender Tabelle wird auch deutlich, warum die Kollision zwischen Joystick und Tastatur beim Control-Port 2 (Adresse 56320) nicht so gravierend ist: Wer drückt schon im Normalfall die angegebenen Tastenkombinationen.

Controlport 1	**(56321)**	**Controlport 2**	**(56320)**
Leertaste	— Feuer	CTRL und J	— Feuer
CTRL-Taste	— links	CTRL und D	— links
1	— oben	CTRL und Cursor rechts	— oben
2	— rechts	CTRL und G	— rechts
<	— unten	CTRL und A	— unten

4/7.1.9

Letzte Zeilennummer nach Programmabbruch

Haben Sie nach einem Programmabbruch durch die STOP-Taste versehentlich den Bildschirm gelöscht und wollen wissen, welche Zeile die Meldung „BREAK IN ...“ enthalten hat, können Sie diese mit

```
PRINT PEEK(57) + 256*PEEK(58)
```

erfahren.

4/7.1.10

Ein einfaches MERGE

Zwei Basic-Programme können Sie sehr einfach verbinden, wenn Sie wie folgt vorgehen:

- — erstes Programm laden
- — PRINT PEEK(43), PEEK(44)
- — Ergebnis merken (Basic-Anfang)
- — POKE 43,(PEEK(45) + 256*PEEK(46)-2) AND 255
- — POKE 44,(PEEK(45) + 256*PEEK(46)-2) /256
  (Anfangszeiger auf Ende des Programmes setzen)
- — zweites Programm laden
- — POKE 43, (erste gemerkte Zahl)
- — POKE 44, (zweite gemerkte Zahl)
- — SYS 42291 (Programme verbinden)

Achtung: das zweite Programm muß höhere Zeilennummern haben als das erste Programm.

4/7.1.11

Programm retten

Folgende Anweisungen funktionieren natürlich nur, wenn Sie ein Programm versehentlich mit NEW oder einem Reset gelöscht haben. In diesem Fall steht das Programm noch im Speicher und nur einige Zeiger wurden durch den Rechner „umgebogen“ (Alle Programmzeilen sind jeweils mit RETURN abzuschließen).

```
— POKE 46, PEEK(56)-1
— POKE 45, PEEK(55)+247
— CLR
  (Zeiger auf Start der Variablen an Basic-RAM-Ende)
— POKE PEEK(44)*256 + PEEK(43)+1, PEEK(44)
— 63999
— FOR I=PEEK(44)*256+PEEK(43) TO PEEK(46)+256 + PEEK(45):IF PEEK(I)
  OR PEEK(I+1) OR PEEK(I+2) THEN NEXT
  (! Abkürzungen verwenden ! / Suche nach Programmende:
  drei aufeinanderfolgende 0-Bytes)
— POKE 45, (I+3)AND 255
— POKE 46, (I+3)/256
— CLR
  (Neues Programmende setzen)
```

4/7.1.12

RAM-Bereich speichern / laden

Zum Speichern von Grafiken (sofern diese im zugriffsfähigen RAM liegen) und Maschinenprogrammen ist es oft sinnvoll, RAM-Bereiche direkt zu speichern. Dies geht mit:

```
SYS 57812 "Name",8
POKE 193, (Anfangsadresse Low Byte)
POKE 194, (Anfangsadresse High Byte)
POKE 780, 193
POKE 781, (Endadresse Low Byte)
POKE 782, (Endadresse High Byte)
SYS 65496
```

Das Gegenstück dazu ist das Laden an eine gewünschte Adresse:

```
SYS 57812 "Name",8
POKE 780,0
POKE 781, (Zieladresse Low Byte)
POKE 782, (Zieladresse High Byte)
SYS 65493
```

Anschließend kann mit

```
PEEK (782)*256 + PEEK(781)
```

die letzte geladene Adresse+1 abgefragt werden.

4/7.1.13

Software- und Hardware-RESET (Warm- und Kaltstart)

Software-RESET

```
SYS 64738
```

bewirkt einen Sprung auf die Initialisierungsroutine, die beim Einschalten angesprungen wird.

Hardware-RESET

Für einen Hardware-RESET gibt es zwei Möglichkeiten:

a) Pin 1 und Pin 3 am User-Port verbinden (GND und Reset)
b) Pin 2 und Pin 6 am seriellen Bus verbinden (erreichbar am Floppy-Stecker)

Nicht vergessen: Basic-Zeiger sind im Urzustand und müssen für das aktuelle Programm gesetzt werden.

4/7.1.14

Speichern auf Kassettenrekorder und Floppy verhindern

Wer seine Programme vor unerlaubter Weitergabe schützen möchte, der probiere folgendes:

Kassettenrekorder

POKE 0,7	gaukelt dem Benutzer nur das Abspeichern vor. Probieren Sie LOAD!
POKE 0,47	Normalzustand

Floppy

POKE 819,253	ergibt Missing File-Name
POKE 818,253	
POKE 808,225	STOP-Taste ausschalten
POKE 819,253	ergibt den, den Assembler-Programmierern hinlänglich bekannten,
POKE 818,34	Befehl
POKE 808,225	BIF (Branch in Forest)
POKE 818,237	Normal-Modus
POKE 819,245	
POKE 808,237	

Bemerkung: in 818/819 steht der SAVE-Vektor, in 808/809 steht der STOP-Vektor

4/7.1.15

Synthetische Steuerzeichen (Finkeln)

Steuerzeichen sind in normalem Text eingebettete Zeichen, die nicht ausgegeben werden, sondern eine andere Aktion veranlassen, wie z.B. Bildschirm löschen (inverses Herz) oder die Cursor-Bewegungen.

Neben der normalen Tastenfunktion (z.B. durch CLR/HOME) können diese Zeichen aber auch synthetisch erzeugt werden. Für ein vorhandenes Steuerzeichen könnte man dies wie folgt durchführen:

Eingabe	Wirkung
?" "	Leere Zeichenreihe ausdrucken. Beachten Sie: Sie sind nicht mehr im „Texteingabemodus".
DEL-Taste	hinteres »"«-Zeichen löschen
RVS-ON-Taste	Invers-Modus direkt einschalten
Q	Steuerzeichen für Cursor nach hinten eingeben
RVS OFF-Taste	Invers-Modus direkt ausschalten
RETURN-Taste	Übergabe an den Rechner

Probieren Sie es auch mit der Eingabe „S" (Cursor in linke, obere Bildschirmecke / HOME).

Hier eine Liste von synthetischen Steuerzeichen, die auch mit PRINT CHR$(X) bewirkt werden können:

Zeichen	CHR$-Code	Bedeutung
H	8	Umschaltung Groß/Kleinschrift durch C=/SHIFT wird gesperrt
I	9	Gegenteil zu H
M	13	RETURN; Rest der Zeile wird nicht mehr ausgegeben
N	14	Umschalten auf Kleinschrift
T	20	ersetzt DEL-Taste
SHIFT + M	141	SHIFT RETURN z.B. Drucken des Textes ab diesem Zeichen in der nächsten Zeile
SHIFT + M R		Ausführen der Steuerzeichen auch im Listing, in diesem Fall REVERSE ON. Probieren Sie auch andere Steuerzeichen
SHIFT + N	142	Umschalten auf Großbuchstaben
SHIFT + T	148	ersetzt INST-Taste

4/7.1.16

Nützliche und hilfreiche PEEKs, POKEs und SYS-Aufrufe

Cursor an/aus

```
POKE 204, 0                 an
POKE 207,0:POKE204,1        aus
```

Cursorgeschwindigkeit ändern

```
POKE 56325,X            X — gewünschte Geschwindigkeit
       1 — schnell,..., 255 langsam
       0 — Bildschirmausgabe verlangsamen
      52 — normaler Zustand, seltene Werte zwischen 51 und 59 in unregelmäßigen
           Abständen
```

Achtung: Da sich die Interrupt-Geschwindigkeit ändern kann, geht die TI-Uhr falsch.

Dauerfunktion für Tasten

```
POKE 650, x
      x = 128 : alle Tasten
      x = 0   : normal (Cursor- und Leertaste)
      x = 64  : alle Tasten ohne Dauerfunktion
```

Directory ohne Programmverlust laden

```
POKE 44, PEEK (46)+1
LOAD "$",8
POKE 43,1 : POKE 44,8
```

Farbwechsel unter dem Cursor

PEEK (647) gibt die Farbe unter dem Cursor an. Dementsprechend kann Sie mit

```
POKE 647, x          x = 0,...,15
```

geändert werden.

FILE-Parameter in der Zero-Page

```
PEEK (183)          — Länge des Filenamens
PEEK (184)          — aktuelle Filenummer
PEEK (185)          — aktuelle Sekundär-Adresse
PEEK (186)          — aktuelle Primäradresse
PEEK (187, 188)     — Zeiger auf Dateinamen
```

Freier Speicherplatz

Die korrekte Ausgabe des freien Speicherplatzes (Basic) erhalten Sie mit

```
PRINT FRE(X)+65536, wenn FRE(X) negativ
```

INVERS-Modus

```
POKE 199,1          ein
POKE 199,0          aus
```

Länge der momentanen Bildschirmzeile

Diese kann mit PEEK (213) abgefragt werden.

Listschutz ein/aus

```
POKE 775,200    (ein)
POKE 775,167    (aus)
```

LIST während Programmablauf

```
100 POKE 631,82
110 POKE 632,69
120 POKE 633,212
130 POKE 634,13
140 POKE 198,4
150 LIST
```

Bedeutungen der Adressen:

631 - 633 : Abkürzung für Basic-Befehl RETURN
634 : RETURN-Taste
198 : Anzahl gültiger Zeichen im Tastaturpuffer

(vergleiche auch ausführlicheres Beispiel unter DELETE). Das Programmstück ist als Unterprogramm aufzurufen, obwohl kein RETURN am Ende steht. Dies wird durch das Programm selbst übernommen. Sinnvoll ist der Einsatz beim Programmtest.

LOAD ERROR umgehen (Kassettenrekorder)

```
POKE 45,PEEK(831)
POKE 46,PEEK(832)
CLR
```

Durch SAVE wird ein Programm zweimal auf Kassette gespeichert. Wird beim Laden eine gewisse Zahl von Fehlern überschritten, tritt ein LOAD ERROR auf, auch wenn das zuerst geladene Programm fehlerfrei ist (z.B. Beschädigung des Bandes, dort wo sich die Kopie befindet). Aber versuchen kann man es immer (vielleicht ist auch ein Fehler in einer Stringkonstanten).

Programm zerstören

Um ein Programm gegen unbefugte Benutzer — etwas — abzusichern, kann man nach einem falsch beantworteten Passwort das Programm sehr leicht mit

```
POKE 776,1
```

zerstören, indem der Vektor für die Basic-Befehlsadresse umgesetzt wird.

Andere Möglichkeit:

```
SYS 64738 (Systemreset)
```

(Siehe auch Listschutz)

Statusabfrage

```
PRINT ST
```

oder

```
PRINT PEEK(144)
```

gibt den Gerätestatus aus.

RUN / STOP aus/ein

```
POKE 788,52    (aus)
POKE 788,49    (ein)
```

RUN/STOP-RESTORE aus/ein

```
POKE 808,225      (aus)
POKE 808,237      (ein)
```

STOP-Taste sperren (andere Möglichkeit)

```
POKE 830, 169          LDA
POKE 831, 1            #1
POKE 832, 96           RTS
POKE 808,62
POKE 809,3             (830)
```

Das Betriebssystem setzt durch die Stop-Taste das Zero-Flag in einem kleinen Programm. Durch obige Routine wird ein Unterprogramm in den Kassettenpuffer geschrieben, das immer das Zero-Flag löscht. Die Speicherzellen 808 und 809 zeigen auf die Betriebssystem-Routine.

STOP-Taste freigeben

```
POKE 808,237        Anfangsadresse
POKE 809,246        Betriebssystem-Routine eintragen
```

Gedrückte Taste

```
PEEK (203)
```

Characterwert der letzten gedrückten Taste

```
PEEK (215)
```

Umschalten auf Groß/Kleinschrift

```
PRINT CHR$(14)    — Groß/Kleinschrift
PRINT CHR$(142)   — Normalmodus
```

Warteschleifen

Neben dem bekannten

```
10 GET A$ : IF A$ = " " THEN 10
```

gibt es noch die Version

```
10 GET A$ : ON - (A$=" ") GOTO 10
```

die den Rest der Zeile (bei knappem Platz) auch noch nutzbar macht, weil die IF-Abfrage wegfällt.

Auch möglich

```
WAIT 198,1 : GETA$ (Taste gedrückt)
```

Auf den Joystick wartet

```
WAIT X,127,127     mit  X=56320 für Port 2
                   und  X=56321 für Port 1
```

Zeichenfarbe ändern

Wer die CTRL-Kombinationen umgehen möchte, und damit auch die Steuerzeichen im String, kann auch die Adresse 646 benutzen:

```
POKE 646, (Farbcode)
```

Zeile auf dem Bildschirm löschen

```
POKE 781,Z
SYS  59903
```

Teil 5

Musterprogramme

5/1 Kommerzielle Programme

5/1.3
Tabellenkalkulation

Die Stärke eines Computers, auch eines sogenannten „Homecomputers“, liegt immer noch darin, große Datenmengen übersichtlich und schnell zu ver- und bearbeiten.

Neben Datenverwaltungsprogrammen und dem Bereich Textverarbeitung erfreuen sich gerade auch Tabellenkalkulationsprogramme großer Beliebtheit bei Computerbesitzern, die mit ihrem Rechner nicht nur spielen wollen. Was kann nun ein Tabellenkalkulations-Programm?

Es kann große Zahlenmengen je nach Problemstellung übersichtlich darstellen. Und es kann mit Hilfe von mathematischen Funktionen und Formeln umfangreiche Berechnungen und Auswertungen mit diesem Zahlenmaterial anstellen. Jeder hat sicher schon einmal für Beruf, Schule oder Hausgebrauch eine Tabelle anlegen müssen. Es ging vielleicht um Verkaufsergebnisse verschiedener Mitarbeiter in verschiedenen Gebieten von verschiedenen Produkten. Oder um die Ergebnisse von Versuchen im Physikunterricht. Oder auch einfach nur um die Haushaltsausgaben, die man in Griff kriegen möchte.

Alle diese Aufgabenstellungen mußten mit Papier, Bleistift und Lineal in stundenlanger Arbeit erledigt werden. Mit einem Tabellenkalkulationsprogramm gelingt dies nicht nur schneller und übersichtlicher, die einmal erstellten Arbeitsblätter (Arbeitsformulare/Rechenvorschriften) können auch jederzeit ohne viel Aufwand geändert und ergänzt werden.

Wir wollen Ihnen im Rahmen dieses Buches ein solches Tabellenkalkulationsprogramm als Musterlisting vorstellen.

5/1.3.1

Das Konzept von ALIPLAN

ALIPLAN wurde nicht mit dem Anspruch geschrieben, bekannten und „berühmten“ Programmen dieser Art Konkurrenz zu machen. Es ist ein Programm, dessen Leistungsfähigkeit auf das Wesentliche konzentriert wurde, um den Programmieraufwand klein und die Bedienungsfreundlichkeit hochzuhalten. Wenn Sie sich näher mit dem Programm befaßt haben, so wird es für Sie ein leichtes sein, eigene Ergänzungen in das vorgestellte Listing einzubauen.

Folgende Punkte unterscheiden ALIPLAN von käuflichen Tabellenkalkulationsprogrammen:

— Es bietet alle Möglichkeiten, die ein Tabellenkalkulations-Programm **mindestens** für ein komfortables Arbeiten aufweisen muß.

— Es ist menügesteuert und deshalb auch wirklich leicht zu bedienen, auch für den Computeranfänger.

— Es berücksichtigt die begrenzten Möglichkeiten der Single-Floppy des C 64 und steht deshalb vollständig im Speicher. Es ist also kein Nachladen während der Bearbeitung notwendig!

Die Menüsteuerung erspart es Ihnen, sich erst Dutzende von Kommandos zu merken, bevor Sie mit dem Programm arbeiten können. Sie können, nachdem Sie sich diese Anleitung einmal sorgfältig durchgelesen haben, sofort loslegen.

5/1.3.2

Die Tastaturverwendung

Hier folgen ein paar Hinweise für einige wichtige Tasten. Die **RETURN**-Taste ist eine der wichtigsten Tasten (wie immer) bei der Arbeit ALIPLAN. Nach jeder

Eingabe von Daten und Kommandos muß die RETURN-Taste in der Regel gedrückt werden. Sie sagt dem Rechner, daß eine Daten- oder Kommandoeingabe beendet ist und das Programm fortgesetzt werden kann.

Die **Leer(SPACE)**-Taste dient dazu, Leerstellen darzustellen. Das Programm ALIPLAN fordert Sie aber auch an verschiedenen Stellen auf, die Leertaste zu drücken, um das Programm fortzusetzen. Dies ist z.B. nach einer Fehlermeldung möglich.

Die **SHIFT**-Taste dient dazu, von der Kleinschreibung auf die Großschreibung umzuschalten. Aber auch viele Sonderzeichen, wie z.B. „#“ können mit der SHIFT-Taste erreicht werden.

Mit der **COMMODORE(C)**-Taste neben der **SHIFT**-Taste auf der linken Seite erreichen Sie einige Grafikzeichen. Z.B. durchgezogene Linien zum Unterstreichen.

Die **INST/DEL**-Taste dient dazu, bei der Eingabe von Daten oder Kommandos, falsche Zeichen zu löschen.

Mit den **Cursor**-Tasten bewegen Sie den Zellzeiger auf Ihrem Arbeitsblatt. Diese Tasten sind auch in Verbindung mit der **SHIFT**-Taste zu bedienen.

Rechts neben der eigentlichen Tastatur befinden sich die **Funktions**-Tasten. Sie werden im Programm ALIPLAN als Kommando-Tasten benutzt, wobei die Tasten mit den geraden Nummern (2, 4, 6, 8) in Verbindung mit der **SHIFT**-Taste gedrückt werden müssen.

5/1.3.3

Laden des Programmes

Bevor Sie das Programm laden, schalten Sie bitte Ihre Geräte in der Reihenfolge Floppy, Drucker, Monitor/Fernseher, Rechner ein. Legen Sie die Programmdiskette ein und tippen Sie dann ein:

LOAD "*",8

und drücken Sie RETURN. Sofern Sie die Programme aus diesem Buch abtippen, muß dafür das Programm 'LOADER' an erster Stelle auf der Diskette stehen. Ansonsten muß ALIPLAN mit LOAD'LOADER',8 gestartet werden. Nach einigen Sekunden meldet Ihr Rechner 'READY'. Tippen Sie jetzt ein:

RUN

und drücken Sie wieder RETURN. Der Bildschirm wird gelöscht, und nach einigen Sekunden werden Sie aufgefordert, die Farben für den Hintergrund, Rahmen und Zeichen zu wählen. Wenn Sie Ihre Wahl getroffen und RETURN gedrückt haben, wird das Hauptprogramm geladen. Nach etwa einer Minute erscheint dann das Arbeitsblatt (siehe Abbildung 5/1.3.4-1) auf Ihrem Bildschirm. Sie sollten nun die Programmdiskette aus dem Floppy-Laufwerk nehmen und eine formatierte Datendiskette einlegen.

5/1.3.4

Das Arbeitsblatt

Stellen Sie sich ein großes leeres Blatt vor, das rasterförmig in 25 Spalten und 80 Zeilen eingeteilt ist. Auf diesem Blatt, dem Arbeitsblatt, befinden sich 25 x 80 = 2000 Zellen, die mit beliebigen Daten gefüllt werden können. Die tatsächlich mögliche Anzahl von gefüllten Zellen ist allerdings von der Breite der Spalten und der Art der Daten (numerisch oder alphanumerisch) abhängig. Der Platz reicht jedoch in jedem Fall für viele hundert Zellen. Da der C 64 jedoch nur 40 Zeichen je Zeile und insgesamt 25 Zeilen auf dem Bildschirm darstellen kann, sehen Sie immer nur einen Ausschnitt Ihres Arbeitsblattes. Wie Sie jedoch trotzdem jede einzelne Zelle erreichen können, wird weiter unten beschrieben. Nachdem Sie das Programm geladen haben, sieht Ihr Bildschirm so aus:

		1	2	3
1		Woche 1	Woche 2	Woche 3
2				
3	Lebensmit	186.35	151.48	198.99
4	Haush.Mit	51.20	33.90	27.30
5	Baby	19.30	21.80	33.54
6	Buecher	7.00	5.00	
7	Zeitungen	3.50	6.00	
8	Schallpl.	19.95		
9	Kleidung		33.00	19.00
10	Computer		61.00	
11	Telefon	45.30		
12	Miete	479.00		
13	Versicher	69.00		15.00
14	Auto	41.54	37.50	25.00
15	Reinigung		9.50	
16	Anschaff.			111.75
17	------------	------------	------------	------------
18	Summe	922.14	359.18	422.58
19				
20	Anteil %	30.38	11.83	13.92

Eintrag Format Kopier Loesch Aendern
Formeln Datei Grafik Druck Ende
*Hausgeld * Z 1 S 1 ALIPLAN

Bild 5/1.3.4-1
Arbeitsblatt von ALIPLAN

Zeilen und Spalten

Oben befindet sich ein Balken, der die Spalten markiert. Am linken Rand sehen Sie einen senkrechten Balken für die Zeilenmarkierung.

Zellzeiger

In der linken oberen Ecke erkennen Sie ein Rechteck, das so breit ist wie eine Spalte. Dies ist der Zellzeiger, der eine Zelle auf dem Arbeitsblatt markiert. Den Zellzeiger können Sie mit den Cursor-Tasten Ihres Computers an jede beliebige Stelle Ihres Arbeitsblattes bewegen. Versuchen Sie es einmal.

Geraten Sie dabei an den Rand des Bildschirmfensters, so wird der gesamte Ausschnitt um eine Spalte oder Zeile nach oben oder unten verschoben. Je nachdem, in welche Richtung Sie den Zellzeiger bewegen.

Zellen

Wenn bisher immer nur allgemein von Zellen gesprochen wurde, so war dies nicht ganz exakt. Wir müssen nämlich von vier Arten von Zellen unterscheiden.

Die ersten drei Arten werden von Ihnen über den Hauptmenüpunkt 'Eintrag' eingegeben. Die meisten Zellen werden sicherlich Zahlen enthalten, mit denen Sie Berechnungen oder Kalkulationen anstellen möchten. Es handelt sich dann um numerische Zellen.

Sollen diese numerische Zellen auch mit einigen Kommastellen dargestellt werden, so spricht man vom Dezimal-Format. Im anderen Fall, wenn nur ganze Zahlen dargestellt werden sollen, spricht man vom Integer-Format.

Da Zahlen aber alleine meist nicht aussagefähig sind, benötigen Sie auch noch Über-/Unterschriften, Bezeichnungen oder Maßeinheiten usw. Diese Zellen, die vornehmlich aus Buchstaben bestehen, aber jedes beliebige Zeichen enthalten können, haben ein alphanumerisches oder Alpha-Format. Die Formatbestimmungen werden unter dem Hauptmenüpunkt 'Format' abgehandelt.

Die letzte Zellenart betrifft das Ergebnis von Berechnung mit Hilfe einer Formel. Diese Zellen, die also die Ergebnisse von Formel-Berechnungen enthalten, werden Ergebniszellen genannt. Das Format dieser Zellen ist immer dezimal!

Funktionstasten-Kommandozeile

Unterhalb des eigentlichen Arbeitsblatt-Ausschnitts sehen Sie eine Leerzeile. Diese Leerzeile wird für die Kommandos der Funktionstasten benötigt. Sie heißt deshalb 'Funktionstasten-Kommandozeile'.

Hauptmenü

Darunter befinden sich zwei Zeilen, die die Funktionen des Hauptmenüs enthalten. Der erste Menüpunkt (oben links: Eintrag) ist durch ein 'Leuchtfeld' gekennzeichnet. Dieses Leuchtfeld kann durch Druck auf die Leertaste bewegt werden. Immer, wenn Sie die Leertaste betätigen, springt das Leuchtfeld auf den nächsten Menüpunkt. Auf diese Art und Weise können Sie eine Hauptmenü-Funktion auswählen. Möchten Sie beispielsweise eine Formel anlegen, so bewegen Sie das Leuchtfeld mit der Leertaste auf den Menüpunkt 'Formeln' und können hier wiederum verschiedene Funktionen auswählen. Doch dazu später mehr.

Name des Arbeitsblattes

Unterhalb des Hauptmenüs befinden sich zwei wichtige Informationen. Ganz links stehen zwei » * «. Hier erscheint der Name eines Arbeitsblattes, das Sie von der Diskette geladen haben.

Zellzeiger-Koordinaten

Rechts daneben stehen zwei Zahlen, die durch einen Schrägstrich getrennt sind. Es handelt sich um die Koordinaten des Zellzeigers. Bewegen Sie den Zellzeiger einmal mit den Cursor-Tasten. Sie werden sehen, daß automatisch die Koordinaten korrigiert werden. Die Reihenfolge ist Zeile/Spalte! So können Sie mit einem Blick die genaue Position des Zellzeigers feststellen.

5/1.3.5

Das Menü und die Bearbeitungsmöglichkeiten

Alle notwendigen Eingaben, Anweisungen oder Informationen, die die Funktionen des Hauptmenüs betreffen, werden entweder in den vier Zeilen unterhalb des Arbeitsblattes, oder aber in einem Anweisungsfenster dargestellt. Das Anweisungsfenster legt sich sozusagen über Ihr Arbeitsblatt, solange Sie mit der Abwicklung einer Funktion beschäftigt sind.

5/1.3.5.1

Eintrag von Zellen

Das Füllen von Zellen erfolgt über die Eingabezeile, die sich unterhalb des eigentlichen Arbeitsblattes befindet. Der Zellzeiger muß vorher an die Stelle gebracht werden, an der die Daten auf dem Arbeitsblatt erscheinen sollen. Wenn Sie mit der Leertaste das Leuchtfeld auf den Menüpunkt 'Eintrag' gebracht haben, drücken Sie bitte die RETURN-Taste. Nun erscheint im Anweisungsbereich ein Lichtpunkt unterhalb von 'Eintrag'.

Außerdem gibt das Programm an, welches Format die angewählte Zelle hat. Geben Sie nun die Daten ein. Sollen alphanumerische Eingaben nicht linksbündig dargestellt werden, wo drücken Sie gleichzeitig die Leertaste und SHIFT so oft, wie eingerückt werden soll.

Nachdem Sie die RETURN-Taste gedrückt haben, wird die Eingabe entsprechend dem Zellenformat an die Stelle auf Ihrem Arbeitsblatt gebracht, auf der sich der Zellzeiger befindet. Sie können maximal 11 Zeichen je Zelle eingeben. Befand sich der Zellzeiger auf einer Zelle, die bereits mit Daten gefüllt war, so erscheint die Meldung 'Zelle belegt'. Das Programm schützt also Ihre Eintragungen vor unbeabsichtigtem Überschreiben. Das gleiche geschieht auch, wenn Sie versuchen, in eine

Ergebniszelle Daten zu schreiben. Wollen Sie Ihre Dateneingabe beenden, so drücken Sie ausschließlich RETURN und Sie befinden sich wieder im Hauptmenü.

5/1.3.5.2

Format

Das Programm unterscheidet drei Zellenformate: Alphanumerisch (Alpha), Dezimal und Integer. Für jede Zelle auf Ihrem Arbeitsblatt kann eines dieser Formate bestimmt werden. Eine Ausnahme bilden nur Ergebniszellen, die grundsätzlich das Format 'Dezimal' haben. Die Formatierung kann zeilen-, spalten- und zellenweise erfolgen. Außerdem können Sie für numerische Zellen die Anzahl der Dezimalstellen, das Füllzeichen und das führende Zeichen bestimmen. Die Spaltenbreite ist frei zwischen 3 und 11 Stellen wählbar. Wenn Sie das Programm geladen und gestartet haben, sind folgende Parameter vorgegeben:

— Spaltenbreite	9
— Anzahl Dezimalstellen	2
— Füllzeichen	Leerzeichen
— Führendes Zeichen	Leerzeichen
— Zellenformat	Dezimal

Nachdem Sie das Leuchtfeld auf den Menüpunkt 'Format' gebracht und RETURN gedrückt haben, erscheint im Anweisungsbereich der Aufruf für folgende Parameter:

— Spaltenbreite
— Anzahl Dezimalstellen
— Füllzeichen
— Führendes Zeichen

Drücken Sie für Füllzeichen und Führendes Zeichen jeweils die Taste für das gewünschte Zeichen. Ein Leerzeichen müssen Sie mit SHIFT eingeben. Bei der Spaltenbreite ist zu berücksichtigen, daß numerische Daten grundsätzlich mit einem Vorzeichen dargestellt werden. Es werden maximal 6 Dezimalstellen formatiert dar-

gestellt. Enthält eine Zelle mehr Zeichen als formatiert dargestellt werden können, so wird die maximale Anzahl linksbündig angezeigt.

Beispiel: Sie haben eine Spaltenbreite von 8 Zeichen gewählt. Eine numerische Zelle enthält aber z.B. 11 Zeichen (—1234567.90). Diese Zelle wird nun linksbündig mit 8 Stellen angezeigt: (—1234567.).

Wenn Sie die obigen Eingaben mit RETURN überspringen, werden die zuletzt gewählten Parameter beibehalten. Anschließend wird die spaltenweise Formatierung aufgerufen. Geben Sie Spaltennummer (1-25) und Format (0-2) an und schließen Sie die Eingabe mit RETURN ab.

Bei der nachfolgenden zeilenweisen Formatierung verfahren Sie ebenso.

Bei der zellenweisen Formatierung müssen die Koordinaten für Zeile und Spalte angegeben werden. Auch diese Abfragen können mit RETURN ohne Auswirkung übersprungen werden. Beachten Sie bitte, daß immer die letzte Formatanweisung eine vorherige überschreibt. Formatieren Sie etwa Spalte 1 Dezimal und anschließend Zeile 1 Integer, so lautet das Format für die Zelle Zeile 1/Spalte 1 Integer, obwohl die Spalte 1 das Format Dezimal erhalten hat! Alle Parameter-Aufrufe erfolgen nacheinander und können grundsätzlich mit RETURN ohne Wirkung übersprungen werden. Alle Formatparameter können jederzeit geändert werden.

5/1.3.5.3

Kopieren

Alle Zeilen und Spalten sowie jede Zelle kann beliebig oft kopiert werden. Es gelten jedoch folgende Einschränkungen:

— Ergebniszellen werden nicht kopiert.
— Bereits gefüllte Zellen werden beim Kopieren geschützt und nicht überschrieben.

Kopieren und Zeilen und Spalten

Nachdem Sie aus dem Hauptmenü 'Kopieren' gewählt und hier die Funktion 'Zeile' oder 'Spalte', werden Sie aufgefordert, die Nummer der zu kopierenden Zeile/Spalte und anschließend die Ziel-Zeile bzw. Spalte einzugeben. Schließen Sie die Eingabe mit RETURN ab.

Kopieren von Zellen

Jeder Zelleninhalt kann durch einfaches Bewegen des Zellzeigers in eine andere Zelle kopiert werden. Bringen Sie dazu den Zellzeiger zuerst auf die zu kopierende Zelle. Nun bewegen Sie das Leuchtfeld auf den Menüpunkt 'Kopieren' und drücken RETURN. Wählen Sie aus dem Untermenü nun '1' für 'Zelle kopieren'.

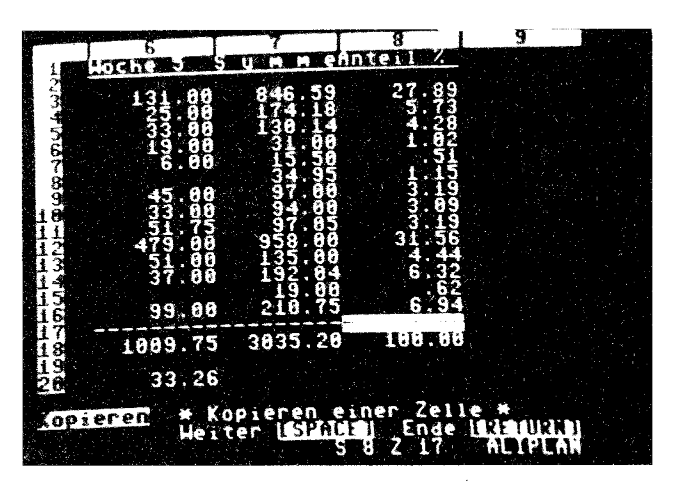


Bild 5/1.3.5.3
Kopieren mit ALIPLAN

Die Kopierfunktion muß nun noch durch Drücken der Leertaste eingeschaltet werden. Dies erkennen Sie an der reversen Darstellung von '* Kopieren einer Zelle *'. Wenn Sie jetzt den Zellzeiger mit den Cursortasten bewegen, wird der Zelleninhalt in die angewählten Zellen kopiert. Dies geschieht so lange, bis Sie die RETURN-Taste gedrückt haben. Dann befinden Sie sich wieder im Untermenü 'Kopieren'.

5/1.3.5.4

Löschen

Bewegen Sie das Leuchtfeld auf den Menüpunkt 'Lösch' und drücken Sie RETURN. Nun werden Sie aufgefordert, durch Drücken der Tasten 1 bis 4 eine Löschfunktion anzuwählen:

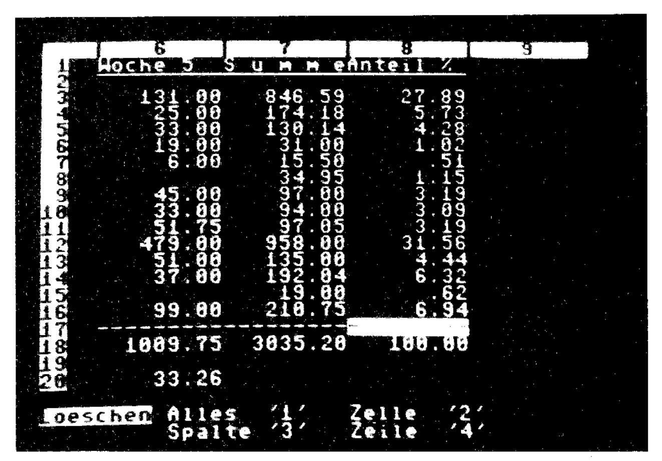


Bild 5/1.3.5.4-1
Das Menü zum Löschen

(1) Alles:

Das Programm fragt zuerst, ob Ihre Entscheidung auch richtig war. Antworten Sie mit 'j', so wird das gesamte Arbeitsblatt gelöscht und alle Formatparameter auf den Einschaltzustand gesetzt. 'n' führt Sie zum Hauptmenü zurück.

(2) Zelle

Bringen Sie den Zellzeiger zuerst auf die zu löschende Zelle und das Leuchtfeld auf den Menüpunkt 'Lösch'. Drücken Sie dann RETURN und wählen Sie dann '2' für 'Zelle löschen'. Nachdem Sie nun wieder RETURN gedrückt haben, ist die Zelle gelöscht.

(3) Zeile
(4) Spalte:

Geben Sie die zu löschende Zeile oder Spalte an. Bei irrtümlicher Wahl von '3' oder '4' gelangen Sie durch RETURN wieder in das Hauptmenü.

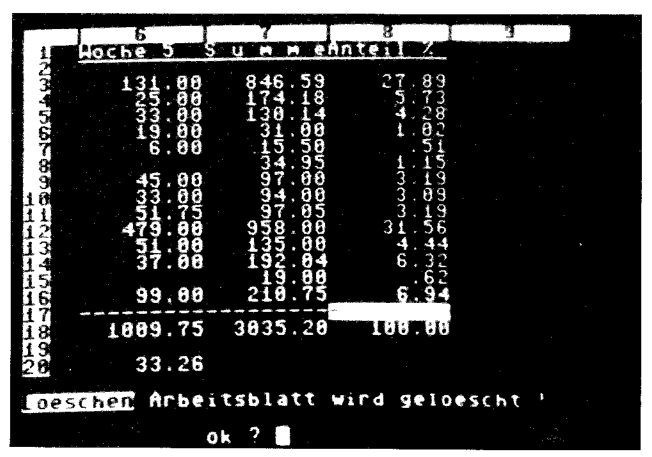


Bild 5/1.3.5.4-2
Sicherheitsprüfung beim Löschen

5/1.3.5.5

Ändern

Bewegen Sie den Zellzeiger auf die zu ändernde Zelle und das Leuchtfeld auf den Menüpunkt 'Ändern' und drücken Sie RETURN. Das Programm zeigt Ihnen jetzt den unformatierten Inhalt der angewählten Zelle. Geben Sie nun den neuen Zellinhalt ein und drücken Sie RETURN. Wenn Sie ausschließlich RETURN drücken, gelangen Sie ohne Änderung der angewählten Zelle in das Hauptmenü zurück.

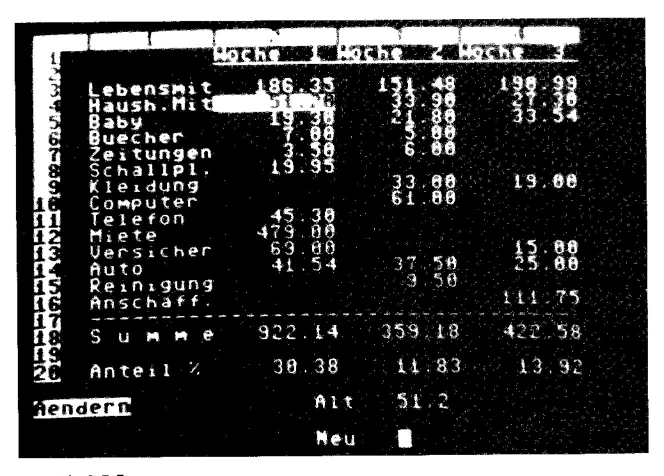


Bild 5/1.3.5.5
Ändern des Eintrages einer Zelle

5/1.3.5.6

Formeln

Die wichtigste Aufgabe eines Tabellenkalkulationsprogrammes ist es, mit Hilfe geeigneter Formeln Rechenoperationen mit den Daten des Arbeitsblattes auszuführen. ALIPLAN bietet Ihnen die Möglichkeit, alle numerischen Werte Ihres Arbeitsblattes mit insgesamt 13 mathematischen und statistischen Funktionen zu verknüpfen. Außerdem können Sie mit Konstanten und mit Klammern rechnen.

Die Funktionen und ihre Symbole:

+ Addition
— Subtraktion
/ Division
* Multiplikation

Die Funktionen und ihre Symbole:

W Quadratwurzel
E Exponent
S Sinus
C Cosinus
Summe
% Prozent
M Mittelwert
s Standardabweichung
[ ] Konstanten
( ) Klammerrechnung

Die Rechenanweisungen werden in einer Formel zusammengefaßt. Es sind zwei Arten von Wertangaben zu unterscheiden:

— Die numerischen Werte einer Zelle Ihres Arbeitsblattes.
— Konstante Werte.

Jede Zelle Ihres Arbeitsblattes, mit der Sie rechnen wollen, muß bei der Bildung einer Formel mit Ihren Koordinaten aus Zeile und Spalte bezeichnet werden. Die Koordinatenwerte müssen dabei immer zweistellig sein. Wollen Sie z.B. zwei Zellen addieren, so lautet die Formel etwa:

+0101+0102

addiere zum Wert der Zelle in Zeile 1/Spalte 1 den Wert der Zelle aus Zeile 1/Spalte 2. Soll jedoch ein konstanter Wert in Ihre Formel einbezogen werden, so setzen Sie diesen in eckige Klammern: +0101+[23]; addiere zum Wert der Zelle Zeile 1/Spalte 1 den Wert '23'. Jede Formel wird **grundsätzlich** von links nach rechts abgearbeitet, d.h. anders als bei der üblichen Vereinbarung, z.B. Multiplikation und Division vor Addition und Subtraktion auszuführen, rechnet ALIPLAN die Anweisungen immer nacheinander ab. Dieses Prinzip kann durch die Verwendung von runden Klammern aufgehoben werden: +0101*(0102-[3]); multipliziere den Wert der Zelle Zeile 1/Spalte 1 mit der Summe aus der Rechnung: Zelle Zeile 1/Spalte 2 minus '3'. **Mehrfachschachtelung ist hier nicht möglich!**

Vor jedem Wert (Zellenkoordinate oder Konstante) sowie vor jeder runden Klammer muß grundsätzlich ein Funktionssymbol stehen. Jedoch dürfen nie zwei Funktionssymbole unmittelbar hintereinander stehen:

Falsch wäre: +0101—W0102.

In diesem Fall muß der Wurzelteil in runde Klammern gesetzt werden:
+0101—(W0102).

Weitere Beispiele zu diesem Fall:

Falsch: +[22]—C0112 Richtig: +[22]—(C0112).
Falsch: +1221+—1107 Richtig: +1221+(—1107).

Die Formeln dürfen keine Zwischenräume enthalten!

Funktionssymbole	Funktion	Bedeutung/Erklärung
		Die wichtigsten Funktionen im Einzelnen:
%	**Prozent**	Wollen Sie die Prozentdarstellung eines Wertes zu einem Basiswert ermitteln, ist folgende Formel zu bilden: %10031503 wieviel Prozent sind (Wert aus Zelle Zeile 10/Spalte 3) von (Wert aus Zelle Zeile 15/Spalte 3)!
#	Summe	Mit dieser Funktion können Sie die Summe aus den Werten einer Zeile oder einer Spalte Ihres Arbeitsblattes ermitteln: #00070316 summiere die Spalte 7 von Zeile 3 bis Zeile 16. Die ersten vier Ziffern geben an, ob eine Spalte oder eine Zeile summiert werden soll und um welche Zeile oder Spalte es sich handelt. Da es in unserem Beispiel um die Spalte 7 geht, muß der Wert für die Zeile 00 sein. Die Reihenfolge ist auch hier wieder Zeile/Spalte. Die Werte müssen wie immer zweistellig sein! Anschließend geben Sie an, welche Spalten- bzw. Zeilenelemente summiert werden sollen. Soll die **Zeile** 7 summiert werden, so lautet die Formel: #07000316!
M	**Mittelwert**	Mit M wird der statistische, arithmetische Mittelwert der angegebenen Speicherzellen gebildet. Die Syntax von M entspricht der Summenfunktion
s	**Standard-abweichung**	Die Standardabweichung ist eine Funktion aus dem Bereich der Statistik (vgl. entsprechendes Kapitel). Die Syntax von s entspricht der Summenfunktion.

Folgende ungültige Operationen werden vom Programm erkannt und gemeldet:

— Division durch 0,
— negativer Wurzelwert.

Diese Operationen führen nicht zu einem Programmabbruch. Es können insgesamt 150 Formeln angelegt werden.

Alle Formeln werden nacheinander berechnet. Die Ergebniszellen werden unmittelbar nach der Berechnung der entsprechenden Formel gefüllt. Dies bietet die Möglichkeit, Zwischenergebnisse zu bilden und anschließend als Zellenwert in einer weiteren Formel zu verwenden.

Formel anlegen

Bevor Sie eine Formel anlegen, müssen Sie den Zellzeiger auf die Position bringen, an der das Ergebnis dieser Formel erscheinen soll. Nun bewegen Sie das Leuchtfeld auf den Menüpunkt 'Formeln' und drücken RETURN. Anschließend wählen Sie »1« für 'Formel anlegen'.

	1	2	3	4
1		Woche 1	Woche 2	Woche 3
2				
3	Lebensmit	186.35	151.48	190.99
4	Haush.Mit	51.20	33.90	27.30
5	Baby	19.30	21.80	33.54
6	Buecher	7.00	5.00	
7	Zeitungen	3.50	6.00	
8	Schallpl.	19.95		
9	Kleidung		33.00	19.00
10	Computer		61.00	
11	Telefon	45.30		
12	Miete	479.00		
13	Versicher	69.00		15.00
14	Auto	41.54	37.50	25.00
15	Reinigung		9.50	
16	Anschaff.			111.75
17	----------	----------	----------	----------
18	Summe	922.1[illegible]	359.18	422.58
19				
20	Anteil %	30.38	11.83	13.92

Formel Nr 56

Formelzelle belegt Taste druecken !

Bild 5/1.3.5.6-1
Formeln bei ALIPLAN

Nun erscheint die Nummer der nächsten Formel und darunter der Lichtpunkt für die Eingabe. Der Pfeil nach oben '↑' markiert die maximale Länge eines Formeleintrages (110 Zeichen). Geben Sie nun die gewünschte Formel ein und schließen Sie die Eingabe mit RETURN ab. Das Programm prüft nun automatisch auf formale Richtigkeit der Formel. Ein Fehler wird mit einem Kommentar angezeigt und die fehlerhafte Stelle mit einem Lichtpunkt markiert. Drücken Sie in diesem Fall die Leertaste und geben Sie die Formel neu ein.

Fehlermeldungen

Funkt-code fehlt/ungültig

Der Funktionscode fehlt oder ist ungültig. Sie haben vergessen, einer Konstanten oder einer runden Klammer des Funktionssymbol, zu schreiben, oder aber das Symbol ist ungültig. Der Funktionscode muß auch vor dem ersten Wert (Zellenkoordinate oder Konstante) stehen. Üblicherweise wird dies das Pluszeichen '+' sein.

Ungültige Zelle

Sie haben einen Zeilenwert angegeben, der kleiner als 1 oder größer als 80 ist, oder einen Spaltenwert, der kleiner als 1 oder größer als 25 ist. Diese Fehlermeldung erfolgt auch, wenn auf einen Funktionscode unmittelbar ein weiterer Code folgt.

Klammer ']' fehlt

Es wurde vergessen, eine Konstante mit der eckigen Klammer zu markieren.

Klammer ')' fehlt

Es wurde vergessen, einen Klammerausdruck mit der runden Klammer abzuschließen.

Ungültiger Wert

Diese Fehlermeldung bezieht sich auf die Angabe der Spalte oder Zeile in den Funktionen '#', Standardabweichung 's' und Mittelwert 'M'. Entweder sind beide Werte größer als 0 (#03110115) oder beide Werte sind gleich 0 (s00000115). Ebenso werden Zeilenwerte kleiner 1 oder größer 80 sowie Spaltenwerte kleiner 1 oder größer 25 gemeldet.

Nach **korrekter** Eingabe einer Formel erscheint das Hauptmenü. Sie können nun den Zellzeiger auf die nächste gewünschte Ergebniszelle setzen und nach RETURN sofort die nächste Formel eingeben. Wollen sie die Formeleingabe beenden, so drücken Sie ausschließlich die RETURN-Taste. Jede Ergebniszelle wird vor Überschreiben durch eine weitere Formel geschützt.

Formeln lesen/ändern

Bringen Sie zuerst wieder den Zellzeiger auf die Ergebniszelle der Formel, die Sie lesen oder ändern wollen. Danach wählen Sie den Menüpunkt 'Formeln' und hier '2' für 'Lesen/Ändern'. Nachdem Sie RETURN gedrückt haben, zeigt das Programm Formel und Formelnummer an.

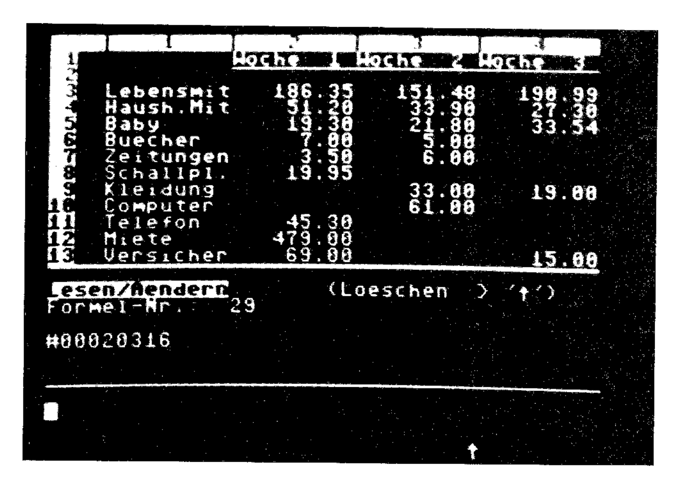


Bild 5/1.3.5.6-2
Formeln lesen und ändern

Soll nicht geändert werden, so drücken Sie einfach wieder RETURN. Im anderen Falle geben Sie nun die neue Formel vollständig ein. Sollte die angewählte Zelle keine Ergebniszelle sein, so macht das Programm Sie hierauf aufmerksam. Durch Drücken einer Taste kommen Sie wieder in das Hauptmenü.

Formel einfügen

Da alle Formeln in der eingegebenen Reihenfolge abgearbeitet werden, kann es notwendig werden, eine Formel an einer Stelle einzufügen. Wählen Sie eine '3' im Untermenü 'Formeln', nachdem Sie wieder den Zellzeiger auf die gewünschte Position gebracht haben. Nach Drücken von RETURN erscheint die Aufforderung, die Nummer der Formel, vor der eingefügt werden soll, anzugeben. Die Nummer erfahren Sie durch den Untermenüpunkt 'Lesen/Ändern'. Geben Sie eine Nummer an, die noch nicht belegt war, so kehrt das Programm wieder zur Abfrage der Formelnummer zurück. Drücken Sie ausschließlich RETURN, gelangen Sie wieder in das Untermenü zurück. War die Nummer gültig, so werden alle Formelnummern verschoben, und danach fordert das Programm Sie auf, die einzufügenden Formeln einzugeben.

Formel kopieren

Das Programm bietet Ihnen die Möglichkeit, einmal erstellte Formeln auf Knopfdruck zeilen- oder spaltenweise zu kopieren. Hierbei wird jedoch nicht einfach der Inhalt einer Formel übernommen, sondern fortgeschrieben.

Beispiel: Sie haben auf Ihrem Arbeitsblatt in den Spalten 1 bis 10 und den Zeilen 1 bis 15 Zahlenkolonnen gebildet, die summiert werden sollen. Das Ergebnis soll jeweils in Zeile 17 unter jeder Zahlenkolonne stehen. Der erste Schritt ist nun, den Zellzeiger auf die Zelle Zeile 17/Spalte 1 zu bringen. Hier geben Sie nun Ihre Formel für diese Spalte ein:

```
# 00010115
```

summiere die Spalte 1 von Zeile 1 bis 15. Anschließend wählen Sie aus dem Untermenü 'Formeln' die Funktion 'Kopieren'. Nachdem Sie die Nummer der zu kopierenden Formel eingegeben haben, erscheint diese Formel und Sie werden aufgefordert, die letzte Zeile oder Spalte anzugeben, bis zu der kopiert werden soll. In unserem Beispiel müssen Sie also die Abfrage der Zeile mit RETURN überspringen und in 'Spalte' '10' eintragen.

Nachdem Sie RETURN gedrückt haben, sind in wenigen Sekunden alle Formeln für die Summierung Ihrer Zahlenkolonnen kopiert. Jeweils in der 17. Zeile erscheint nach der Berechnung die Summe der jeweiligen Spalte. In gleicher Weise können natürlich auch Spaltensummen, Mittelwerte und Standardabweichungen gebildet werden.

Ein weiteres Beispiel soll diese Funktion verdeutlichen: Es sollen die Bruttoumsätze einer Reihe von Waren ermittelt werden. In Spalte 2 stehen die Nettopreise je Stück, in Spalte 3 die verkaufte Menge und in Spalte 4 der Mehrwertsteuersatz. In der Spalte 5 sollen die Bruttoumsätze jeder Warenart stehen.

Zuerst bringen Sie den Zellzeiger auf die Position Zeile 1/Spalte 5 und geben dann die erste Formel ein:

```
+0102*0103+[+0102/(100)*0104]
```

im Klartext: Nettopreis * Menge + Mehrwertsteuer. Nun wählen Sie wieder '4' für 'Kopieren' und geben die Nummer obiger Formel an. Anschließend tippen Sie in 'Zeile' die letzte Zeile Ihrer Zahlenkolonne ein. Nach RETURN werden alle Formeln für Ihr Arbeitsblatt automatisch erstellt.

5/1.3.5.7

Diskettenoperationen

Jedes einmal erstellte Arbeitsblatt kann als Datei auf Diskette gespeichert und jederzeit wieder geladen werden. Darüberhinaus bietet ALIPLAN die Möglichkeit, eine Datei zu löschen und das Inhaltsverzeichnis einer Diskette zu lesen.

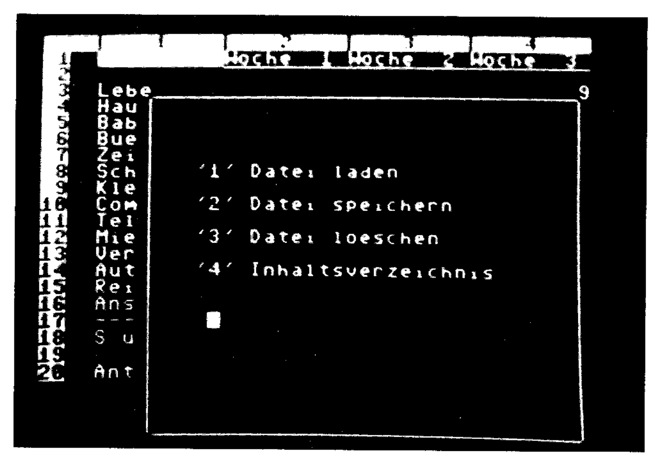


Bild 5/1.3.5.7-1
Das Menü der Diskettenoperationen wird in das Arbeitsblatt eingebettet

Datei (Arbeitsblatt) laden

Das Programm lädt normalerweise nur mit ALIPLAN erstellte Daten. Theoretisch ist es jedoch möglich, versehentlich andere, nicht mit ALIPLAN erstellte Dateien zu laden. Dies würde jedoch in der Regel zu einem Programmabbruch führen. Wollen Sie ein Arbeitsblatt laden, nachdem Sie ein anderes bearbeitet haben, sollten Sie das Arbeitsblatt zuerst vollständig löschen. Wählen Sie den Hauptmenüpunkt 'Datei' und hier '1' für 'Programm laden'. Nach Druck auf die RETURN-Taste werden Sie aufgefordert, den Dateinamen, unter dem das Arbeitsblatt abgelegt worden ist, anzugeben.

Kann die gewünschte Datei nicht gefunden werden, erfolgt die Meldung 'file not found'. Entweder war dann der Dateiname falsch (achten Sie auf Groß-/Kleinschreibung) oder Sie haben eine falsche Diskette eingelegt. Das Programm meldet auch alle anderen Disketten-Fehler. Schauen Sie in diesen Fällen bitte in Ihr Floppy-Handbuch.

Waren die Eingaben korrekt, erscheint das Arbeitsblatt nach einigen Sekunden auf Ihrem Bildschirm.

Datei (Arbeitsblatt) speichern

Wählen Sie im Untermenü Datei '2' für 'Programm speichern'. Sie werden nun aufgefordert, den Dateinnamen, unter dem das Arbeitsblatt gespeichert werden soll, anzugeben. Anschließend geben Sie bitte die letzte Zeile und die letzte Spalte an, die gespeichert werden soll. Der Sinn dieser Maßnahme liegt darin, Speicherplatz auf der Diskette und Zeit beim Speichern und Laden zu sparen, da ja nicht immer alle vorhandenen Zellen auch belegt werden.

Nun wird das Arbeitsblatt mit allen Formeln und Format-Parametern auf Diskette abgelegt. Beachten Sie jedoch bitte, daß auch die Formatbestimmungen nur für den angegebenen Bereich (letzte Zeile/Spalte) gespeichert werden. Sollten Sie also die Absicht haben, in Ihrem Arbeitsblatt später Zeilen oder Spalten ergänzen zu wollen, so wählen Sie den zu speichernden Bereich bitte entsprechend.

Ein gespeichertes Arbeitsblatt kann auch jederzeit überschrieben werden. In diesem Fall, wenn also der angegebene Dateiname bereits vorhanden ist, macht das Programm Sie darauf aufmerksam. Drücken Sie 'j', wenn die vorhandene Datei überschrieben werden soll. Wollen Sie dies verhindern, so drücken Sie 'n' für nein.

Datei (Arbeitsblatt) löschen

Wählen Sie '3' für 'Datei löschen' und drücken Sie RETURN. Geben Sie dann den Dateinamen an. Beachten Sie bitte, daß normalerweise nur Dateien, die mit ALI-PLAN erstellt wurden, gelöscht werden können. Das Programm fragt aus Sicherheitsgründen noch einmal, ob Ihre Entscheidung richtig war. Antworten Sie mit 'j', wird diese Datei gelöscht. Antworten Sie mit 'n', wenn die Löschung verhindert werden soll. Ist die angegebene Datei auf der eingelegten Diskette nicht vorhanden, meldet das Programm 'keine Datei'.

Inhaltsverzeichnis

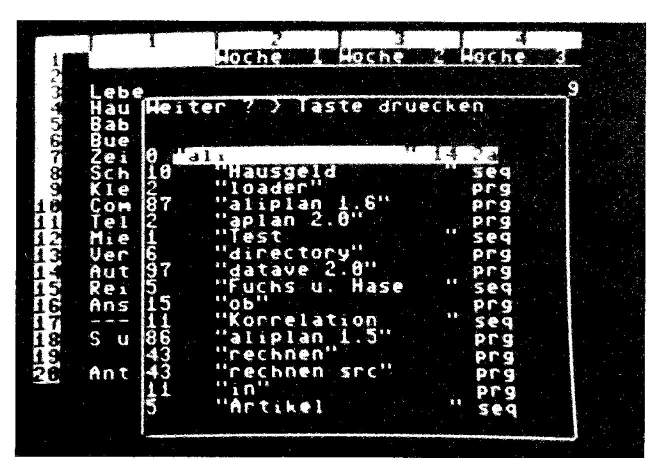


Bild 5/1.3.5.7-2
Anzeige des Directorys in einem Bildschirmfenster über dem Arbeitsblatt

Wählen Sie '4' für 'Inhaltsverzeichnis'. Das Programm zeigt Ihnen nun das komplette Inhaltsverzeichnis der eingelegten Diskette an. Reicht der Platz im Anweisungsfenster nicht, um alle Dateien anzuzeigen, so wartet das Programm, bis Sie eine Taste drücken. Nun wird die Anzeige fortgesetzt bzw. beendet.

5/1.3.5.8

Grafik

ALIPLAN stellt Balkendiagramme von beliebigen Spaltenausschnitten. Bewegen Sie das Leuchtfeld auf den Hauptmenüpunkt 'Grafik' und drücken Sie RETURN. Die Werte-Spalte enthält die numerischen Werte, die grafisch dargestellt werden sollen. Unter Anzeige-Spalte wird die Spalte Ihres Arbeitsblattes verstanden, die in

Klarschrift zu den Grafikbalken dargestellt werden soll. Dies kann die gleiche Spalte wie die Werte-Spalte sein, aber auch jede andere Spalte. Darüber hinaus müssen Sie noch den Spaltenausschnitt wählen (von Zeile bis Zeile). Es können aber nur maximal 20 Zeilen gewählt werden.

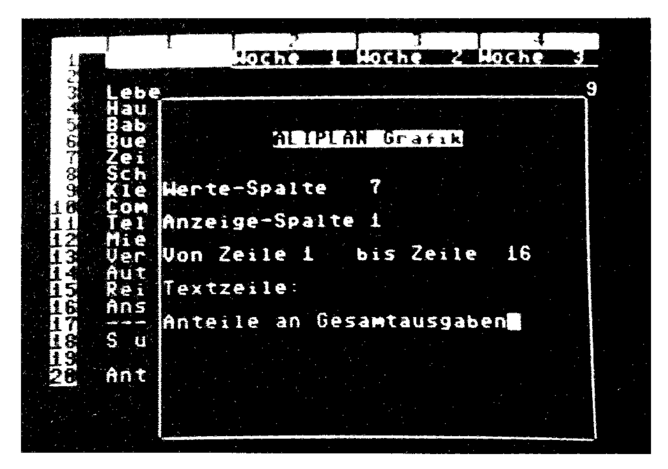


Bild 5/1.3.5.8-1
Erfassen der Parameter zur Grafik-Ausgabe

Beispiel: Sie haben ein umfangreiches Arbeitsblatt Ihrer Haushaltsausgaben erstellt. Die Spalte 1 enthält die Bezeichnungen der Ausgabeposten (Lebensmittel, Miete etc.). Die Spalte 13 enthält die Summe der Positionen für das aufgelaufene Jahr. Geben Sie nun '13' als Werte-Spalte und '1' als Anzeige-Spalte ein. Anschließend noch von Zeile '1' bis Zeile '20'.

Dann können Sie noch eine Textzeile in Ihre Grafik einfügen, die die Darstellung kommentiert. Dies ist sinnvoll, wenn Sie eine Hardcopy der Grafik wünschen. Nach diesen Eintragungen erscheinen nun links die Bezeichnungen der Ausgabeposten und rechts daneben die den Summen der Einzelposten entsprechenden Grafikbalken.

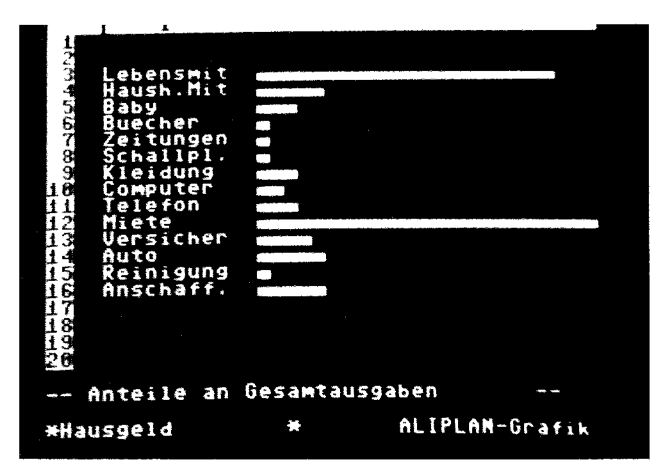


Bild 5/1.3.5.8-2
Ausgabe einer Balkengrafik bei ALIPLAN

Wünschen Sie eine Hardcopy Ihrer Grafik auf dem Drucker, so drücken Sie die Taste ‘f3’ (Funktionstaste 3). Mit RETURN gelangen Sie wieder in das Hauptmenü zurück.

5/1.3.5.9

Drucken

Sie können Ihr Arbeitsblatt zur Dokumentation ausdrucken. Das Druckformat wird automatisch bestimmt. Es werden so viele Spalten wie möglich nebeneinander gedruckt. Besteht Ihr Arbeitsblatt aus mehr Spalten, als nebeneinander gedruckt werden können, so bildet das Programm Blöcke, die untereinander ausgedruckt werden. Auf eine Druckseite werden nur vollständige Blöcke gebracht. D.h. es wird ein Seitenvorschub ausgelöst, wenn ein Block nicht mehr vollständig auf eine Seite passen würde.

Eine Ausnahme bilden nur Arbeitsblätter, die aus mehr als 57 Zeilen bestehen. Nachdem Sie das Leuchtfeld auf den Hauptmenü-Punkt 'Druck' gebracht und RETURN gedrückt haben, fragt das Programm, ob der Drucker eingeschaltet ist. Prüfen Sie dies und antworten Sie dann mit 'j'. Nun geben Sie den Umfang des auszudruckenden Arbeitsblattes an. Anschließend müssen Sie die Richtigkeit Ihrer Angaben bestätigen. Drücken Sie 'j', wenn die Angaben richtig sind, und 'n' wenn Sie die Angaben korrigieren möchten. Wenn der Druckvorgang beendet ist, kehrt das Programm in das Hauptmenü zurück.

5/1.3.6

Kommandos der Funktionstasten

Alle nachfolgenden Funktionen stehen Ihnen immer dann zur Verfügung, wenn Sie sich im Hauptmenü befinden. Alle Ein- und Ausgaben erfolgen über die Funktionstasten-Kommandozeile zwischen Arbeitsblatt und Hauptmenü.

Funktionstaste	Bedeutung und Erklärung
f1	**Arbeitsblatt berechnen** ALIPLAN berechnet das Arbeitsblatt auf Tastendruck vollständig neu. Die ungültigen Operationen 'Division durch 0' und 'Wurzel aus einem negativen Wert' werden mit der Ergebniszelle, in der sie auftraten, gemeldet. Drücken Sie in diesem Fall die Leertaste, um das Programm fortzusetzen.
f3	**Hardcopy** ALIPLAN bietet die Möglichkeit, den Bildschirminhalt als sogenannte Hardcopy auf Ihrem Drucker auszudrucken. Diese Funktion steht Ihnen auch innerhalb des Hauptmenü-Punktes 'Grafik' zur Verfügung. Achten Sie bitte darauf, daß der Drucker eingeschaltet ist, wenn Sie diese Funktion aufrufen. Um die Lesbarkeit des Ausdruckes nicht zu beeinträchtigen, wurde auf inverse Darstellung verzichtet.

Funktionstaste	Bedeutung und Erklärung
f5	**Freie Zeilen** Das Programm gibt Ihnen die Anzahl der noch freien Zellen in Abhängigkeit von der gewählten Spaltenbreite an. Diese Funktion kann bei gefülltem Arbeitsblatt einige Sekunden in Anspruch nehmen. Beachten Sie auch, daß die Angaben nur Näherungswerte sind.
f7	**Zellzeiger auf Home-Position setzen** Unter Home-Position wird die linke obere Ecke des Arbeitsblattes verstanden. Auf Tastendruck wird der Zellzeiger von jeder beliebigen Stelle Ihres Arbeitsblattes auf die HOME-Position Ihres aktuellen Arbeitsblatt-Ausschnittes gesetzt. Drücken Sie die Taste f7 nun noch einmal, so kehrt der Zellzeiger auf die Position Zeile 1/Spalte 1! zurück. Somit haben Sie jederzeit die Möglichkeit, auf Tastendruck zur Ausgangsposition zu gelangen.
f2 **f4**	**Einfügen einer Zeile** **Einfügen einer Spalte** An jeder Stelle Ihres Arbeitsblattes kann eine Leerzeile oder Leerspalte eingefügt werden. Die letzte belegte Zeile darf aber nicht größer als 79 und die letzte belegte Spalte nicht größer als 24 sein! Alle Formeln werden nach dieser Operation entsprechend korrigiert. Setzen Sie zuerst den Zellzeiger auf die Zeile oder Spalte, an der eine Leerzeile/Leerspalte eingefügt werden soll. Drücken Sie dann die entsprechende Funktionstaste (f2 oder f4). Das Programm fordert Sie nun auf, die letzte Zelle oder Spalte, die verschoben werden soll, anzugeben. Hier sind wieder die Koordinaten der letzten Zeile/letzen Spalte einzugeben.
f6	**Springe nach Zelle . . .** Da es recht mühsam wäre, auf einem größeren Arbeitsblatt mit den Cursor-Tasten umherzuwandern, haben Sie hier die Möglichkeit, die Koordinaten Ihrer **Zielzelle** einzugeben. Nach RETURN steht der Zellzeiger auf der gewünschten Position, und der Bildschirm zeigt den entsprechenden Arbeitsblatt-Ausschnitt.
f8	**Zelle vollständig anzeigen** Durch die Formatierung der Zellen wird nicht immer jede Zelle vollständig angezeigt. Haben Sie z.B. auf zwei Dezimalstellen formatiert, wird ein Rechenergebnis, das sieben Dezimalstellen aufweist, auch nur mit zwei Dezimalstellen auf dem Arbeitsblatt angezeigt. Durch Drücken der Funktionstaste f8 zeigt Ihnen das Programm den vollständigen Inhalt der Zelle, auf der sich der Zellzeiger befindet.

5/1.3.7

Programmbeschreibung

Wir beginnen mit einer Übersicht über die Verteilung der Zeilennummern

400	Basic-Ende unterhalb der Formatmatrix setzen
590	Formatmatrix löschen
1000 — 1500	Arbeitsblatt auf Bildschirm bringen
3000 — 4000	Eingabe/Füllen von Zeilen
5000 — 6500	Formatbestimmungen
7000 — 7080	Kopieren einer Zelle
9000 — 9430	Löschen von Zelle, Spalte, Zeile und Arbeitsblatt
11000 — 11210	Ändern einer Zelle
13000 — 18180	Eingabe einer Formel
13185 — 13705	Fehlerprüfung einer Formel
14000 — 14100	Ausgabe der Fehlermeldung
14200 — 14210	Formel o.k.! Formelnummer erhöhen
14700 — 14795	Formel ändern oder löschen
14850 — 14890	Formel einfügen
14900 — 14990	Formel kopieren
15000 ff	Dateiverwaltung
15300 — 15450	Datei laden
15500 — 15650	Datei speichern
15700 — 15800	Datei löschen
15900 — 16160	Inhaltsverzeichnis Diskette
16900 — 16960	Fehlermeldungen Floppy
17000 — 17020	Arbeitsblatt berechnen
19000 — 20040	Arbeitsblatt ausdrucken
21000 — 21111	Programm beenden
22000 — 23500	Grafik erstellen
30000 — 30115	Menüzeilen aufbauen
30200 — 30219	Steuertasten abfragen
30220 — 30248	Zellencursor bewegen
30265 — 30320	Hauptmenü-Auswahl
30400 — 30435	Cursor auf Homeposition
30450 — 30465	Zeile einfügen
30500 — 30530	Spalte einfügen
30600 — 30640	Springe nach Zeile

31000 — 32010	Formel neu referenzieren (Werteverweise)
33000 — 33060	Scrollen nach rechts
33100 — 33160	Scrollen nach links
33200 — 33261	Scrollen nach oben
33300 — 33360	Scrollen nach unten
40500 — 40530	Cursor-Zelle nach Absprung anzeigen
51000 — 51360	Arbeitsblatt anzeigen

Das Programm besteht aus ca. 22 KByte Basiccode und etwa 3,6 KByte Maschinencode.

Ein wesentliches Problem bestand darin, möglichst viele Programmfunktionen mit möglichst geringem Speicherbedarf zu realisieren, da die Größe des zur Verfügung stehenden Arbeitsblattes eben vom restlichen Speicherraum abhängt. Insbesondere sollte auch ein Nachladen von Programmteilen während des Programmlaufs vermieden werden. Es wurde deshalb auf jegliche Kommentierung im Programm verzichtet und darüber hinaus versucht, so viele Befehle wie möglich in einer Programmzeile unterzubringen. Konsequente Benutzung von Schleifen statt direkter, linearer Programmierung war deshalb ebenfalls angesagt.

Insgesamt wurde versucht, die Gesichtspunkte Speicherbedarf, Größe des Arbeitsblattes, Bedienungsfreundlichkeit (z.B. durch Menütechnik) und Ausführungsgeschwindigkeit zu optimieren. Um eine akzeptable Ausführungsgeschwindigkeit zu erreichen, mußten ‘geschwindigkeitsempfindliche’ Routinen in Assembler programmiert werden. Hier ist insbesondere der Neuaufbau des Bildschirms und die Routine zur Auswertung und Berechnung der Formeln zu nennen.

Für eine komfortable Bildschirmgestaltung wurden weitere zahlreiche Assemblerroutinen eingefügt. Alle Maschinenroutinen werden mit dem Befehl SYS, gefolgt von der Adresse und eventuell zu übergebender Parameter, mit Kommata getrennt, aufgerufen. Eine genaue Übersicht über alle verwendeten Maschinenroutinen folgt später.

Nachfolgend beziehen wir uns erst einmal auf den Basic-Teil des Programms. Da das Programm modular strukturiert ist, läßt sich fast jeder Funktion ein bestimmter, geschlossener Programmbereich (siehe auch Übersicht nach Programmzeilen) zuordnen.

Variablen initialisieren

Der Bereich unterhalb der Programmzeile 1000 dient im wesentlichen der Initialisierung wichtiger Variablen. Insbesondere auch Stringvariablen für den Menüaufbau und Fehlermeldungen etc. Hier werden auch die Adressen der Maschinenroutinen Variablen zugewiesen, um ihre Handhabung im Programm zu erleichtern (540 — 550).

Arbeitsblatt initialisieren

In den Zeilen 1000 bis 1500 wird das Arbeitsblatt in der Grundstellung (Spalte 1 und Zeile 1 in der oberen linken Ecke) und die Hauptmenüauswahl auf den Bildschirm gebracht.

Eingabe der Zellen

Bei Eingabe von Zellen (3000 bis 4000) wird geprüft, ob die Zelle bereits belegt ist und gegebenenfalls eine Fehlermeldung ausgegeben. Dies soll vor unbeabsichtigtem Überschreiben bereits gefüllter Zellen schützen. Weiterhin wird geprüft, ob es sich um eine Formelzelle handelt, die das Ergebnis einer Formel bereitstellt. Auch in diesem Fall wird ein Überschreiben verhindert (3050 bis 3083). Ist die Zelle noch leer, wird das Format (dezimal, numerisch, rechtsbündig, alphanumerisch) ermittelt und die Eingabe freigegeben (3084 bis 4000).

Die Informationen über das Zellenformat befinden sich in einer Formmatrix. Hier steht für jede Zeile ein Code, der das jeweilige Format definiert. Alle Zellen haben ursprünglich das Format 'dezimal' (wird in Zeile 590 initialisiert). Die Änderung erfolgt mit der Funktion 'Format'. Angemerkt werden soll noch, daß alle Zellenvariablen Strings sind und ihr Format erst bei der Ausgabe auf den Bildschirm bzw. auf den Drucker bestimmt wird.

Formatbestimmungen

In den Zeilen 5000 bis 6500 werden die Formatattribute aller Zellen bzw. Spalten und Zeilen festgelegt. Die Parameter 'Anzahl Dezimalstellen', 'Füllzeichen' und 'Führendes Zeichen' werden der Print-Using-Routine übergeben und können nur global, d.h. für alle Zellen, bestimmt werden.

Die anderen Parameter werden in die Formatmatrix eingetragen. Lediglich die Formelzellen behalten immer grundsätzlich das Format 'dezimal'. Es wird also auf das Vorhandensein einer Formelzelle geprüft (6080 — 6090) und die Formatierung dieser Zelle gegebenenfalls unterbunden.

Zellen editieren

Der Bereich von Zeile 7000 bis Zeile 11210 beinhaltet das Kopieren und Löschen von Zellen, Spalten und Zeilen sowie das Ändern einer Zelle. Der Inhalt aller entsprechenden Zellenvariablen wird gelöscht oder kopiert — ohne allerdings die Formatmatrix zu ändern. Zu erwähnen ist noch, daß hier zwar der Ergebniswert einer Formelzelle geändert oder gelöscht werden kann, nicht jedoch die Formel selbst. Diese Funktionen finden Sie unter Hauptmenüpunkt 'Formeln'.

Formeln

Der Bereich von Zeile 13000 bis 14990 behandelt das Erstellen und Editieren einer Formel. Alle Formeln werden in separaten Formelstring-Variablen gespeichert. Zur

Identifikation beginnt jeder Formelstring mit den Koordinaten (Zeile, Spalte) für die Ergebniszelle. Außerdem wird nach erfolgter Fehlerprüfung in der Formatmatrix die Formelnummer+2 an der entsprechenden Position eingetragen (14205). Auf diese Weise enthält jede Formel den Bezug zur Ergebniszelle und die Formatmatrix den Bezug zur Formelvariablen.

Um zu verhindern, daß das Programm bei der Auswertung einer Formel wegen Syntaxfehlern aussteigt, wurde eine Routine zur Fehlerprüfung integriert (13185 bis 13705). Sofort nach Eingabe einer Formel wird diese Routine aufgerufen. Bei einem Fehler wird die Fehlerposition innerhalb der Formel invers angezeigt und zur Korrektur aufgefordert. Außerdem wird noch eine Fehlermeldung ausgegeben (14000 bis 14100). Erst nach einer fehlerfreien Eingabe erzeugt ALIPLAN in den Zeilen 14200 bis 14205 die eigentliche Formelvariable und trägt in die Formatmatrix die Formelnummer ein.

Vor der beabsichtigten Löschung oder Änderung einer Formel muß der Zellcursor auf die entsprechende Formelzelle gesetzt werden. Handelt es sich nicht um eine Formelzelle, wird dies angezeigt. Die Zeilen 14782 und 14783 bewirken ein Blinken der Fehlermeldung. Da alle Formeln in der Reihenfolge ihres Index' abgearbeitet werden, kann es notwendig sein, Formeln an irgendeiner Stelle einzufügen. Diese Routine in den Zeilen 14850 bis 14890) prüft in 14850, ob die maximale Anzahl zulässiger Formeln erreicht ist. In einer Schleife (14860 bis 14875) wird dann der Index der höheren Nummern um eine Einheit erhöht und auch die Formatmatrix entsprechend korrigiert.

ALIPLAN kopiert eine Formel, indem nach Möglichkeit die Werteverweise (Zeile, Spalte) neu errechnet werden. Nachdem die zu kopierende Formelnummer (14910) und der Bereich (14935) eingegeben wurden, isoliert das Programm die fortzuschreibenden Parameter und springt in einer Schleife ab 14960 eine Routine an (31000 bis 32010), die die Neureferenzierung der Werteverweise durchführt. Die Neuberechnung des Arbeitsblattes erfolgt durch eine umfangreiche Assemblerroutine (17000 — 17020), die über die Funktionstasten aufgerufen wird (30207).

Diese Routine übergibt an die Speicherstellen 852, 853 und 854 einen Fehlercode sowie die Koordinaten (Zeile, Spalte) der betroffenen Formelzelle im Falle einer Division durch Null oder eines negativen Wurzelarguments. Die Maschinenroutine wurde so eingestellt, daß bei diesen beiden Fehlern kein Programmabbruch erfolgt.

Diskettenoperationen

ALIPLAN bietet eine Reihe von Diskettenoperationen an. Beim Speichern oder Laden eines Arbeitsblattes reicht es aber nicht, nur die Zelleninformationen zu transferieren, es müssen auch alle Formelstrings und alle Formatparameter (Formatmatrix) übergeben werden.

In Zeile 15505 wird der zu speichernde Bereich erfragt. ALIPLAN speichert (und lädt) nur die Informationen für den spezifizierten Bereich, um Zeit und Speicher-

platz auf der Diskette zu sparen. Die Daten werden in eine sequentielle Datei übernommen (15550 bis 15650). Das Laden eines Arbeitsblattes erfolgt entsprechend in den Zeilen 15300 bis 15450. Eine Maschinenroutine gestattet ab Zeile 15900 das Lesen des Inhaltsverzeichnisses der Arbeitsdiskette.

Arbeitsblatt ausdrucken

Eine Druckroutine (19000 bis 20040) ermöglicht es, das gesamte Arbeitsblatt oder einen beliebigen Ausschnitt auszudrucken. Nachdem die Bereichsparameter eingegeben wurden (19000 bis 19230) gestaltet das Programm automatisch das Druckformat. In den Zeilen 19300 bis 19420 wird die Anzahl der darzustellenden Spalten und Zeilen je Seite ermittelt. Danach erfolgt die eigentliche Druckroutine (19500 bis 19700). Da mit dem Befehl CMD gearbeitet wird (19350), kann die Print-Using-Routine für einen formatgerechten Ausdruck benutzt werden (19550).

Nach jedem Seitenvorschub wird der Kopf neu gedruckt (20000 bis 20040).

Grafik

Die Grafikroutine in den Zeilen 22000 bis 23500 setzt eine Wertespalte des Arbeitsblattes in eine einfache Balkengrafik um. In den Zeilen 22290 bis 22310 wird der höchste Wert für die Dimensionierung ermittelt. Anschließend wird die Balkengrafik erstellt. Das Programm hält nun an, bis entweder RETURN oder die Funktionstaste für eine Hardcopy gedrückt wurde (22410 bis 23500).

Cursorbewegung, Hauptmenüauswahl, Funktionstastenbefehle

Ab Zeile 30000 bis 30320 erwartet das Programm einen Tastenbefehl zur Steuerung des Zellzeigers, zur Auswahl einer Hauptmenüfunktion oder eines Funktionstastenbefehls. Zuerst wird die Cursorbewegung erfragt (30202 bis 30206) und gegebenenfalls in die Routine zur Cursorbewegung verzweigt (30220 bis 30248). Hier prüft das Programm jeweils, ob die Grenze des aktuellen Arbeitsblattausschnittes überschritten wird. In diesem Fall verzweigt ALIPLAN in die entsprechende Routine zum Scrollen und Ergänzen des Bildschirms (33000 bis 33360). Für jede Bewegungsrichtung sind entsprechende Routinen vorhanden bzw. werden Maschinenroutinen aufgerufen. Nach dem horizontalen Scrollen ergänzt eine schnelle Assemblerroutine das Arbeitsblatt um die neu hinzugefügte Spalte.

Nach jeder Bewegung des Zellzeigers muß die inverse Darstellung der Ausgangsposition umgekehrt (40500 bis 40530) und die Zielposition wiederum invers dargestellt werden (40550 bis 40585).

Gleichzeitig werden die Parameter für die aktuellen Bildschirmpositionen und Variablenindizes korrigiert.

Anschließend fragt das Programm die Funktionstasten ab und ruft die entsprechenden Funktionen bzw. Routinen auf. Beim Einfügen von Zeilen oder Spalten in das Arbeitsblatt (30450 bis 30530) ist es notwendig, die Formeln entsprechend zu kor-

rigieren (dazu wird jeweils die schon bekannte Routine zur Neureferenzierung der Werteverweise, ab Zeile 31000, aufgerufen), um die Einträge in die Formelmatrix zu verschieben.

In den Zeilen 30265 bis 30320 ruft das Programm die schon besprochenen Hauptmenüfunktionen auf. Dies geschieht in der Weise, daß nach jedem Betätigen der Leertaste der Menüzeiger (Variable kz) auf die jeweilige Funktion zeigt (inverse Darstellung). Wird nun RETURN gedrückt, verzweigt das Programm mit dem Befehl ON GOSUB in die gewählten Routinen (30265).

Arbeitsblatt aufbauen

Die Routine ab Zeile 51000 baut das Arbeitsblatt bei Bedarf neu auf. Hier findet die Assemblerroutine zur Ergänzung einer Spalte Anwendung, die in Zeile 51170 aufgerufen wird, nachdem die entsprechenden Parameter an diese Routine übergeben wurden. Die Zeile 51360 bewirkt die Ergänzung des Arbeitsblattes um die Zeilen und Spaltennummer sowie der Hauptmenüauswahl-Anzeige.

Von ALIPLAN verwendete Assemblerroutinen		
Name/Funktion	**Adresse**	**Syntax**
Scrollen nach links	49152	Adr., von Zeile, bis Zeile, Anzahl, von Spalte, nach Spalte
Scrollen nach rechts	49424	Adr., von Zeile, bis Zeile, Anzahl, von Spalte, nach Spalte
Scrollen nach oben	49716	Adr., von Zeile, bis Zeile, von Spalte, bis Spalte
Scrollen nach unten	49934	Adr., von Zeile, bis Zeile, von Spalte, nach Spalte
Print an Cursorposition	49690	Adr., Spalte, Zeile, Ausdruck/Variable
Print-Using	50155	Adr., Spalte, Zeile, Ausdruck/Variable Nur numerische Ausdrücke oder Variablen. Zu übergebender Parameter: Integer oder Dezimal Adresse+201 (0 oder 1) Länge des auszugebenden Strings Adresse+202 (0 — 10) Dezimalstellen Adresse+203 Füllzeichen Adresse+204 (ASCII-Code) Führendes Zeichen Adresse+310 (ACSII-Code)

Von ALIPLAN verwendete Assemblerroutinen		
Name/Funktion	**Adresse**	**Syntax**
Hardcopy	50360	Adresse
Clear Bildschirm- bereich	50451	Adr., von Zeile, bis Zeile, von Spalte, bis Spalte
Input	50622	Adr., Spalte, Zeile, Länge, Typ, Variable Type = 0 numerisch oder 1 String Füllzeichen Adresse+433 (ASCII-Code)
Directory	51233	Adresse, Spalte, Zeile
(alle vorgenannten Routinen sind im Programm 'ali 2.0' zusammengefaßt)		
'aliplan 2.5' bringt ausgewählte Spalten des Arbeitsblat- tes auf den Bildschirm	51460	Adresse Zu übergebender Parameter: 141 von Spalte 142 von Zeile 830 von Element low 831 von Element high 832 von Element low 833 von Element high
'rechnen' Auswertung und Berechnung der Formeln	51763	Adresse

5/1.3.8

Variablenliste

a	Maschinenprogramm nachladen
as$	Formelstring retten
an	Anzahl Stellen/Spalte
as	Anzahl Spalten/Bildschirm
az	Anzahl Zeilen/Bildschirm
bs	bis Spalte
bz	bis Zeile
c	Spalte (Zellenindex)
cl	Adresse für Clear-Routine
d$	Druckstring
dl$	Flag: Datei löschen
dn$	Dateiname
dz	Anzahl Dezimalstellen
el	Fehlermeldung Floppy
e2$	Fehlermeldung Floppy
e3$	Fehlermeldung Floppy
e4$	Fehlermeldung Floppy
e5$	Datei überschreiben?
f$ (x)	Formelstring
ff	Flag für Format
fr$ (x)	Fehlermeldungen
fz	Anzahl Formeln
fz$	Füllzeichen
in	Adresse der Input-Routine
k$ (x)	Hauptmenü-Bezeichnungen
ko$	zu kopierende Zellenvariable
kz	Nummer der Hauptmenü-Bezeichnung
l$	Leerstring für Zellcursor
m0 — m9	Rechenvariablen für verschiedene Zwecke
m0$ — m9$	Hilfsstrings für verschiedene Zwecke
mx	Startadresse der Formatmatrix
p	Flag für verschiedene Zwecke
pp	Formatcode/Rechenvariable
pr	Adresse der Print-Routine

ps	Spaltenposition auf Bildschirm
pz	Zeilenposition auf Bildschirm
pu	Adresse der Print-Using-Routine
r	Zeile (Zellenindex)
r$	geänderte Zellenvariable
r$ (x, y)	Zellenvariable
rr	numerischer Wert einer Zellenvariablen
r0 — r9	Rechenvariablen für verschiedene Zwecke
r0$ — r9$	Formelstring editieren
sh	Erste Spaltennummer für aktuellen Arbeitsblattausschnitt
sl	Adresse der Routine Scrollen nach links
so	Adresse der Routine Scrollen nach oben
sp	aktuelle Spalte
sr	Adresse der Routine Scrollen nach rechts
ss	Differenz Beginn Arbeitsblatt — erste tatsächliche Spalte
su	Adresse der Routine Scrollen nach unten
t	Formel anlegen, lesen/ändern, einfügen, kopieren
t$	allgemeine get-Variable
t%	Flag: Hauptmenüaufruf und verschiedene Zwecke
tt%	Löschen Spalte, Zeile, Zelle, Arbeitsblatt
uz$	führendes Zeichen (Zellenformatierung)
vs	von Spalte

Das Programmm befindet sich auf der Grundwerksdiskette

5/2 Mathematisch-technisch-wissenschaftliche Programme

Mathematisch-technisch-wissenschaftliche Programme stellen den Einsatzbereich dar, weshalb Rechner überhaupt entwickelt wurden: Umfangreiche Berechnungen schneller als der Mensch auszuführen.

Auch wir wollen diesem Thema im Rahmen der Musterprogramme einigen Raum einräumen. Wir fangen an mit einigen mathematischen Routinen, wie z.B. die Berechnung von Nullstellen bei Gleichungen dritten Grades bzw. Zahlberechnungen mit großer Genauigkeit anhand des Beispiels der Euler'sche Zahl.

Im Rahmen der Statistik werden wir uns auch näher mit der Wahrscheinlichkeitsrechnung am Beispiel von Roulette beschäftigen und eine Reihe statistischer Unterprogramme vorstellen. Im Bereich der technischen Programme werden wir z.B. auf das Thema Statik näher eingehen, aber auch andere technische Unterprogramme vorstellen.

Daß die Physik ein reichhaltiges Anwendungsfeld für den Computer ist, wird sicherlich jedem klar sein. Auch hier werden wir Berechnungen aus den verschiedensten Bereichen vorstellen.

An dieser Stelle möchten wir noch einmal daran erinnern, daß uns jeder Leser Musterprogramme einsenden kann. Dies dürfte besonders für den mathematisch-technisch-wissenschaftlichen Bereich interessant sein, da zu den speziellen Bereichen meist eingehende Sachkenntnis erforderlich ist. Selbstverständlich werden alle Einsendungen entsprechend honoriert.

5/2.1

Mathematische Routinen

5/2.1.1

Nullstellen von Gleichungen bis 3. Grades

(Autor: Klaus Kohl)

Mit dem nachfolgenden Programm können alle realen Nullstellen einer algebraischen Gleichung 3. Grade smit einer Unbekannten ermittelt werden. Unter einer algebraischen Gleichung 3. Grades versteht man eine Gleichung der Form:

$A * x^3 + B * x^2 + C * x + D = 0$

Eine Gleichung n-ten Grades hat maximal n Lösungen. Jedoch kann bei Gleichungen höheren Grades vorkommen, daß es keine reale Lösung gibt oder Nullstellen höherer Ordnung auftreten. Dadurch verringern sich die Anzahl der Lösungen.

Bedienungsanleitung

Es sind alle Variablen von A bis D einzugeben. Sind nur quadratische Gleichungen zu lösen, so ist für A der Wert 0 einzugeben. Nach den Eingaben erfolgt die Ausgabe der gesamten Gleichung und die einzelnen Lösungen. Anschließend kann entweder das Programm beendet oder mit der Eingabe neuer Werte fortgesetzt werden.

Programmaufbau

Bis zu quadratischen Gleichungen wird es den Anwendern leichtfallen, anhand des Programmlistings die Ermittlung der Nullstellen nachzuvollziehen. Bei den kubischen Gleichungen ist das schon schwierig, da hier verschiedene Berechnungsverfahren bei den unterschiedlichen Lösungen verwendet wurden.

Gleichungen dritten Grades besitzen vier Lösungsarten:

a) Es existiert nur eine Nullstelle
b) Es gibt drei verschiedene Nullstellen
c) Zwei Nullstellen fallen zusammen
d) Alle drei Nullstellen sind identisch

Bei der Ermittlung der Lösungen wird deshalb die Gleichung zuerst mittels einer Substitution in die reduzierte Form übergeführt:

Substitution: $z = x + \frac{B}{3 * A}$

Ergebnis: $z^3 + 3 * P * z + 2 * Q = 0$

mit $P = \frac{C}{3 * A} - \frac{B * B}{9 * A * A}$ und $Q = \frac{B * B * B}{27 * A * A * A} - \frac{B * C}{6 * A * B} + \frac{C}{2 * A}$

Diskriminante: $F = P^3 + Q^2$

Anhand der Diskriminante F wird dann festgestellt, welche der Lösungsarten für die eingegebenen Werte zutrifft:

1) $F > 0$: Nur eine Lösung:

$x = z - \frac{B}{3 * A}$ mit $z = \sqrt[3]{-Q + \sqrt[2]{F}} + \sqrt[3]{-Q - \sqrt[2]{F}}$

2) $F = 0$

a) Für $P = Q = 0$: Dreifache Nullstelle

$x_1 = x_2 = x_3 = -\frac{B}{3 * A}$

b) Für $P^3 = -Q^2$: Eine Einfach- und eine Doppelnullstelle

$x_1 = 2 * \sqrt[3]{-Q} - \frac{B}{3 * A}$; $x_2 = x_3 = \sqrt[3]{-Q} - \frac{B}{3 * A}$

3) $F < 0$: Drei verschiedene Lösungen

$R = -\sqrt[2]{|P|}$; $W = \arctan\left(\frac{R^6}{Q^2} - 1\right)$

$x_1 = -2 * R * \cos\left(\frac{W}{3}\right)$

$x_2 = 2 * R * \cos\left(\frac{\pi + W}{3}\right) - \frac{B}{3 * A}$

$x_3 = 2 * R * \cos\left(\frac{\pi - W}{3}\right) - \frac{B}{3 * A}$

Dabei wird in 3) eine trigonometrische Auflösung und bei den anderen Gleichungen die CARDANISCHE FORMELN verwendet:

$$z_1 = u+v; \; z_2 = w_1 * u+w_2 * v; \; z_3 = w_2 * u+w_1 * v$$

$$\text{mit } w_1 = \frac{1}{2}(-1+i * {}^{2}\,3); \; w_2 = \frac{1}{2}(-1 - i * {}^{2}\,3)$$

$$\text{und } u = \sqrt[3]{-Q + \sqrt[2]{F}}; \; v = \sqrt[3]{-Q - \sqrt[2]{F}} \; (\text{für } F \geqq 0)$$

Dabei sind w, u und v weitere Hilfsvariablen. Werden nur reale Lösungen ermittelt, so können diese Formeln auf die bei 1) und 2) verwendeten Gleichungen gekürzt werden.

Schwierigkeiten können aufgrund der begrenzten Rechengenauigkeit des C 64/ C 128 auftreten. Deshalb wurde bei den kubischen Gleichungen bei Fallunterscheidungen nicht abgefragt, ob die Diskriminante den Wert 0 hat. Statt dessen wird geprüft, ob der Betrag der Diskriminante kleiner als 1/1000000 des Wertes von A ist. Um diese Sonderfälle auch noch auszuschließen, müßten weitere Berechnungen durchgeführt werden, was aber den Sinn dieses Programmes überschreiten würde.

Hier noch einige Literaturangaben, mit deren Hilfe dieses Programm erstellt worden ist. Die verwendeten Namen für Variablen sind meistens direkt übernommen worden.

1. NETZ: Formeln der Mathematik
(Carl Hanser Verlag, München Wien)
1. BRONSTEIN/SEMENDJAJEW: Taschenbuch der Mathematik
(Verlag Harri Deutsch, Thun und Frankfurt)
3. BARTSCH: Mathematische Formeln
(Buch- und Zeit-Verlagsgesellschaft mbH, Köln)

Verwendete Variablen:

A, B, C, D	Faktoren
E, Q, P, F, R, W	Diskriminanten und Zwischenvariablen
A$	Eingabe-Variable

Das Programmm befindet sich auf der Grundwerksdiskette

```
40 REM ************************************************
50 REM *** NULLSTELLEN VON GLEICHUNG BIS 3. GRADES  ***
60 REM ************************************************
70 :
80 REM ================== ANFANGSBILD =================
90 :
100 PRINT"****************************************";
110 PRINT"*                                      *";
120 PRINT"* NULLSTELLENBERECHNUNG DER GLEICHUNG: *";
130 PRINT"*                                      *";
140 PRINT"*            3    2    1               *";
150 PRINT"*         A*X +B*X +C*X +D = 0         *";
160 PRINT"*                                      *";
170 PRINT"****************************************"
200 PRINT"EINGABE DER PARAMETER:"
210 OPEN 1,0
220 PRINT"A=";:INPUT#1,A:PRINT
230 PRINT"B=";:INPUT#1,B:PRINT
240 PRINT"C=";:INPUT#1,C:PRINT
250 PRINT"D=";:INPUT#1,D:PRINT
260 CLOSE1
470 :
480 REM ================ KONTROLLAUSGABE ===============
490 :
500 PRINT"DIE GLEICHUNG:"
510 PRINT
520 IF A=0 THEN 550
530 IF A>0 THEN PRINT"+";
540 PRINT A;"*X^3 ";
550 IF B=0 THEN 580
560 IF B>0 THEN PRINT"+";
570 PRINT B;"*X^2 ";
580 IF C=0 THEN 610
590 IF C>0 THEN PRINT"+";
600 PRINT C;"*X ";
610 IF D=0 AND (A<>0 OR B<>0 OR C<>0) THEN 640
620 IF D>0 THEN PRINT"+";
630 PRINT D;
640 PRINT" = 0":PRINT
670 :
680 REM ============== FALLUNTERSCHEIDUNG ==============
690 :
700 IF A<>0 THEN 4000: REM KUBISCHE GLEICHUNG
710 IF B<>0 THEN 3000: REM QUADRATISCHE GLEICHUNG
720 IF C<>0 THEN 2000: REM LINEARE GLEICHUNG
970 :
980 REM ============== GLEICHUNG 0. GRADES =============
990 :
1000 IF D<>0 THEN 1030
1010 PRINT"LAESST FUER X ALLE WERTE ZU."
1020 GOTO 5000
1030 PRINT"IST WIDERSPRUECHLICH (UNLOESBAR).
1040 GOTO 5000
1970 :
```

Listing 5/2.1.1 (1)

```
1980 REM ============== GLEICHUNG 1. GRADES =============
1990 :
2000 PRINT"HAT NUR DIE LOESUNG:"
2010 PRINT"X=";-D/C
2020 GOTO 5000
2970 :
2980 REM ============== GLEICHUNG 2. GRADES =============
2990 :
3000 E=C*C-4*B*D : REM DISKRIMINANTE
3010 IF E=0 THEN 3040
3020 IF E<0 THEN 3070
3030 IF E>0 THEN 3090
3040 PRINT"HAT DIE DOPPELLOESUNG:"
3050 PRINT"X1=X2=";-C/2/B
3060 GOTO 5000
3070 PRINT"BESITZT KEINE LOESUNG."
3080 GOTO 5000
3090 PRINT"HAT ZWEI LOESUNGEN:"
3100 PRINT"X1=";(-C-E^.5)/2/B
3110 PRINT"X2=";(-C+E^.5)/2/B
3120 GOTO 5000
3970 :
3980 REM ============== GLEICHUNG 3. GRADES =============
3990 :
4000 P=C/3/A-B*B/9/A/A
4010 Q=B*B*B/27/A/A/A-B*C/6/A/A+D/2/A
4020 E=B/3/A
4030 F=P*P*P+Q*Q
4040 IF ABS(F)<(1E-6*ABS(A))THEN 4070
4050 IF F<0 THEN 4160
4060 IF F>0 THEN 4230
4070 IF ABS(P) > (1E-6*ABS(A)) THEN 4110
4080 PRINT"HAT DIE DREIFACHE LOESUNG:"
4090 PRINT"X1=X2=X3=";-E
4100 GOTO 5000
4110 PRINT"HAT DREI LOESUNGEN:"
4120 PRINT"X1=";2*SGN(-Q)*ABS(Q)^(1/3)-E
4130 PRINT"X2=X3=";SGN(Q)*ABS(Q)^(1/3)-E
4140 PRINT"X2 UND X3 IST EINE DOPPELTE NULLSTELLE."
4150 GOTO 5000
4160 R=-SQR(ABS(P)):W=π/2:IF Q=0 THEN4180
4170 W=ATN(SQR(R*R*R*R*R*R/Q/Q-1))
4180 PRINT"HAT DREI VERSCHIEDENE LOESUNGEN:"
4190 PRINT"X1=";-2*R*COS(W/3)-E
4200 PRINT"X2=";2*R*COS(π/3+W/3)-E
4210 PRINT"X3=";2*R*COS(π/3-W/3)-E
4220 GOTO 5000
4230 PRINT"HAT NUR EINE LOESUNG:":PRINT"X=";
4240 Z1=-Q+F^.5:Z2=-Q-F^.5
4250 PRINT SGN(Z1)*ABS(Z1)^(1/3)+SGN(Z2)*ABS(Z2)^(1/3)-E
4970 :
4980 REM === ABFRAGE : ENDE ODER WEITERE GLEICHUNGEN ===
4990 :
5000 PRINT"▮WEITERE GLEICHUNGEN (J/N) ?";
5010 GET A$
5020 IF A$="J" THEN 100
5030 IF A$<>"N" THEN 5010
5040 END
```

Listing 5/2.1.1 (2)

5/4 Grafik

5/4.1 Demo-Maker

Autor: Helmut Schröter

Diesmal wollen wir ein Programm zur Erstellung von Demos vorstellen.

5/4.1.1 Die Aufgabe des Demo-Makers

Der Demo-Maker dient zum Erstellen von Vorspannen bzw. von Demos. Er wurde in Assembler geschrieben und bietet die Möglichkeit, einen Text vor einem Grafikbild als Laufschrift zu scrollen und dabei eine Musik spielen zu lassen. Die Laufschrift bewegt sich zudem noch in einem festlegbaren Bereich von oben nach unten und wieder hinauf. Weiterhin ist die Möglichkeit gegeben, einen anderen Zeichensatz für die Laufschrift zu benutzen.

5/4.1.2

Die Menüs

Das Hauptmenü ist folgendermaßen gegliedert:

1. Laufschrifttext eingeben
2. Laufschrift-Menü
3. Grafik laden
4. Musik laden
5. Demo zeigen
6. Demo speichern
7. Directory zeigen
8. Ende

Laufschrift eingeben

Durch diesen Menüpunkt kommt man in einen Editor, der zur Eingabe des Scrolltextes gedacht ist. Es ist darauf zu achten, daß man nicht mit den Cursortasten aus dem unteren Bildschirmrand fährt, da sonst der obere Teil des Textes gelöscht wird. Am Ende des Textes muß die Eingabe eines Klammeraffen erfolgen, um das Textende für die Laufschrift festzulegen. Durch den Druck auf „F1“ wird der Bildschirm im Speicher als Laufschrifttext gesichert.

Laufschrift-Menü

Durch diesen Menüpunkt gelangt man wieder in ein Untermenü, das folgendermaßen gegliedert ist:

1. Springbereich festlegen
2. Zeichensatz laden
3. Farbenblinken ändern
4. Hauptmenü

Im ersten Menüpunkt gelangt man zur Option, die obere und untere Grenze des Springbereiches der Laufschrift festzulegen. Dies geschieht mit Hilfe der Cursortasten oder des Joysticks, der im Port 2 angeschlossen sein muß. Durch die RETURN-Taste bzw. den Feuerknopf wird die Eingabe bestätigt.

Im zweiten Menüpunkt kann man einen neuen Zeichensatz laden. Dazu muß man lediglich den Namen eingeben.

Im dritten Menüpunkt kann man das Farbenblinken der Laufschrift auswählen. Es sind dafür die Farben Blau, Grün, Rot, Grau und Gelb vorgesehen.

Durch den vierten Menüpunkt gelangt man wieder ins Hauptmenü.

Grafik laden

In diesem Untermenü kann man eine Grafik laden, die im Koalapainter-Format auf Diskette steht. Die Steuercodes im Grafiknamen werden dabei außer acht gelassen. Das heißt, daß man nur noch den eigentlichen Bildnamen eintippen muß. Für den Fall, daß das gewünschte Grafikbild nicht im Koala-Format vorhanden ist, muß man dieses mit einem Konvertierungsprogramm bearbeiten. Solche Konvertierungsprogramme sind bereits vielfach erhältlich.

Musik laden

In diesem Untermenü kann man eine Musik, die mit dem Soundmonitor angefertigt wurde, einladen. Diese Musik wird dann während des Demos gespielt. Wenn man keine Musik haben möchte oder nicht im Besitz des Soundmonitors ist, kann man auch auf die Benutzung dieses Unterpunktes verzichten. Der Soundmonitor ist bereits mehrfach erschienen und in der Public Domain oft benutzt worden.

Demo zeigen

Durch die Wahl dieses Punktes wird das Demo so gezeigt, wie es auf Diskette gespeichert werden würde. Man kann so noch einmal das ganze Ergebnis betrachten und eventuelle Änderungen mit Hilfe der vorhergehenden Menüs durchführen. Das Anzeigen des Demos wird durch einen Tastendruck abgebrochen, und man gelangt wieder ins Hauptmenü.

Demo speichern

Das Demo wird durch die Wahl dieses Menüpunktes unter dem Namen „DEMO“ auf Diskette gespeichert und der Demo-Maker durch einen Reset verlassen.

Directory anzeigen

Diese Funktion zeigt das Inhaltsverzeichnis der eingelegten Diskette auf dem Bildschirm.

Ende

Der Demo-Maker wird durch einen Reset des Computers verlassen.

5/4.1.3

Das Erstellen eines Demos

Es wird nun im Folgenden die genaue Vorgehensweise beim Erstellen von Demos erläutert.

Zuerst ist der Laufschrifttext einzugeben. Dies geschieht, wie bereits erwähnt, durch die Anwahl des ersten Hauptmenüpunktes. Nachdem man der Aufforderung nachgekommen ist, eine Taste zu drücken, kann man einen beliebigen Text eingeben, der im Demo über den Bildschirm laufen soll. Es ist darauf zu achten, daß man in der untersten Bildschirmzeile nicht RETURN drückt, nicht mit den CRSR-Tasten nach unten fährt und auch das letzte Zeichen frei läßt. Anderenfalls würde der obere Teil des Laufschrifttextes verlorengehen. Am Schluß des Textes ist ein „Klammeraffe“ einzugeben. Dieses Zeichen befindet sich auf der Tastatur rechts vom P. Wenn der Text fertig eingegeben ist, ist die Taste F1 zu drücken. Dadurch wird der Text in den Textspeicher der Laufschrift kopiert, und man gelangt wieder ins Hauptmenü.

Als nächstes sollte man den Bereich, in dem die Laufschrift springt, nach eigenen Wünschen definieren. Dazu wählt man den zweiten Punkt des Hauptmenüs. Man befindet sich nun im Laufschrift-Menü, das wiederum gegliedert ist. In diesem Menü ist der Punkt 1 anzuwählen. Es erscheint nun eine Anleitung, wie die obere und die untere Grenze gesetzt werden. Dies geschieht mit Hilfe des Joysticks in Port 2 oder durch die Tastatur. Um die obere Grenze zu definieren, bewegt man einfach die Markierung auf dem Bildschirm durch Drücken des Joysticks nach oben bzw. unten auf die gewünschte Position und drückt den Feuerknopf. Das Setzen der unteren Grenze geschieht auf die gleiche Art und Weise. Um die Grenze über die Tastatur zu setzen, ist die Markierung mit den CRSR-Tasten steuerbar. Anstatt des Feuerknopfes ist nun die RETURN-Taste zu betätigen. Nachdem die untere Grenze gesetzt ist, gelangt man automatisch wieder ins Hauptmenü.

Daraufhin kann man einen Zeichensatz für die Laufschrift laden, indem man wiederum ins Laufschrift-Menü und vonda aus in Menüpunkt 2 geht. Es wird dort nach dem Namen des Zeichensatzes gefragt, und nach dessen Eingabe wird dieser in den Speicher geladen. Anschließend befindet man sich wieder im Hauptmenü. Wenn man auf das Laden eines neuen Zeichensatzes verzichten will bzw. muß, wird der sich im Speicher befindende benutzt.

Weiterhin kann man die Farben auswählen, in denen die Laufschrift blinken soll. Dazu muß man ebenfalls wieder ins Laufschrift-Menü und dort den dritten Menü-

punkt wählen. Es wird daraufhin gefragt, in welchen Farbtönen die Laufschrift blinken soll. Die Farbtöne Blau, Grün, Rot, Grau und Gelb sind dafür vorgesehen. Man muß nun nur noch die entsprechende Taste drücken.

Um das Demo schöner gestalten zu können, ist die Funktion „Grafik laden“ enthalten. Die Grafik muß, wie bereits erwähnt, im Koalapainter-Format auf Diskette stehen.

Programmerläuterung

Beim ersten Menüpunkt wird zuerst ein Text auf den Bildschirm gebracht und danach auf einen Tastendruck gewartet. Anschließend wird der Bildschirm gelöscht, das Cursorblinken ausgeschaltet und der Zeichensatz auf Großschrift/Grafik gestellt. Dann wird über die Betriebssystemroutine GETIN jeder Tastendruck mit dem Wert für „F1“ verglichen. Wenn er gleich ist, wird der Bildschirmspeicher in den Textspeicher der Laufschrift kopiert, der Zeichensatz wieder auf Groß-/Kleinschrift gestellt und ins Hauptmenü gesprungen. Wenn er nicht gleich ist, wird das Zeichen auf den Bildschirm geschrieben und GETIN erneut abgefragt.

Beim zweiten Menüpunkt wird auf ein Untermenü verzweigt und gewartet, bis eins angewählt wird. Bei dem Untermenü, in dem der Springbereich festgelegt wird, werden zwei Sprites auf den Bildschirm gebracht, und danach die Werte von GETIN und der Speicherzelle des Joystick-Ports mit den Werten für die Auf- und Abwärtsbewegung und dem Verlassen der Eingabe verglichen. Während dieser Zeit ist zudem noch der Rand mit Hilfe einer Interrupt-Routine ausgeschaltet, damit die Sprites über die gesamte Bildschirmhöhe zu sehen sind. Die Werte, die durch die jeweiligen Spritepositionen definiert sind, werden daraufhin in den Teil, der für das Auf- und Abwärtsbewegen der Laufschrift bestimmt ist, geschrieben. Beim zweiten Untermenü wird ein Zeichensatz in einen unbenutzten Speicherplatz geladen und die ersten 512 Bytes in den Zeichensatzspeicher der Laufschrift kopiert. Dies ist notwendig, da ein Zeichensatz in der Regel länger ist als 512 Bytes, in der Laufschrift allerdings nur diese benötigt werden. Ein direktes Laden an die Speicherplatzstelle des Laufschriftenzeichensatzes würde so einen Teil des hinter dem Zeichensatz abgelegten Laufschrifttextes löschen.

Im dritten Untermenü wird nach den Farbtönen der Laufschrift gefragt, und diese danach an die Speicherstelle der Blinkroutine kopiert.

Das vierte Untermenü springt ohne Änderung von Speicherstellen ins Hauptmenü.

Im dritten Hauptmenüpunkt wird eine Hires-Grafik an die Speicherstelle $2000 geladen und wieder ins Hauptmenü verzweigt.

Beim vierten Punkt wird die Musik an die Speicherstelle $A000 geladen und ebenfalls wieder ins Hauptmenü gesprungen.

Der fünfte Punkt setzt die Musik in Ausgangsposition und die Speicherzelle für das Randmuster, das sich beim Ausschalten des Randes ergibt, auf voll. Daraufhin wird das Bild angezeigt, indem der Hires-Modus angeschaltet wird und die Bilddaten auf den Bildschirm kopiert werden. Im Anschluß wird die Laufschrift gestartet, die wiederum in die Abspielroutine der Musik springt. Zum Schluß wird auf einen Tastendruck gewartet und bei positivem Ergebnis zum Hauptmenü gesprungen.

Durch den sechsten Menüpunkt wird der Speicher von $0801 bis $cfff auf Diskette gespeichert, nachdem eine neue Basiczeile in den Anfang des Speichers geschrieben und der Editor gelöscht wurde. Der Computer macht nach dem Speichern einen Reset.

Der siebte Menüpunkt bringt über die OPEN-Routine im Betriebssystem das Directory der eingelegten Diskette auf den Bildschirm.

Der achte Menüpunkt erzeugt einen Reset des Computers, durch einen Sprung zur Speicherzelle $fce2.

Durch die Anwahl des dritten Punktes des Hauptmenüs gelangt man ins Grafikmenü. Dort muß nur noch der Name der Grafik ohne das reverse Pik-Zeichen und ohne die anderen Parameter angegeben werden. Nachdem die Grafik geladen wurde, befindet man sich wieder im Hauptmenü.

Die Option, eine Musik zu laden, sollte nach Möglichkeit genutzt werden, da das Demo dadurch aufgelockert wird. Um dies zu verwirklichen, muß man lediglich zum vierten Punkt des Hauptmenüs springen und dort den Namen der Soundmonitormusik eingeben. Es ist darauf zu achten, daß die Musik wirklich im Soundmonitorformat auf der Diskette steht und maximal 49 Blocks belegt. Eine Soundmonitormusik ist in der Regel an der Länge von 46 bzw. 49 Blocks zu erkennen und läßt sich in Direktmodus, nachdem sie mit load„Name",8,1 geladen wurde, mit sys 49152 starten.

Mit dem fünften Menüpunkt sollte man sich das ganze Demo vor dem Abspeichern nochmals ansehen und gegebenenfalls noch Änderungen im Springbereich oder im Farbenblinken über die entsprechenden Menüs vornehmen.

Wenn man mit dem Demo zufrieden ist, ist der 6. Menüpunkt zu wählen. Dadurch wird das Demo unter dem Namen „DEMO" auf Diskette gespeichert.

5/4.1.4

Der Basic-Lader

Der Basic-Lader des Demo-Maker ist in zwei Teile aufgeteilt und folgendermaßen zu starten.

Zuerst wird der erste geladen bzw. abgetippt und mit „RUN“ gestartet. Dieser generiert nun den ersten Teil des Demo-Makers auf Diskette. Danach wird mit dem zweiten Teil des Basic-Laders ebenso verfahren. Auf der Diskette befinden sich nun zwei neue Programme mit den Namen „daten1.exe“ und „daten2.exe“. Diese muß man, wenn der Demo-Maker benutzt werden soll, unbedingt laden. Dies geschieht durch:

```
load "daten1.exe",8,1
load "daten2.exe",8,1
```

Danach kann der Demo-Maker mit „RUN“ gestartet werden.

```
1Ø rem demo-maker data lader
2Ø :
3Ø z=1ØØØ:print"zeile:",z:open1,8,15,"iØ":open2,8,2,"daten1.exe,p,w"
35 print#2,chr$(1);chr$(8);
4Ø read z$
5Ø su=Ø:for i=1to67 step2
6Ø w$=mid$(z$,i,2):gosub2ØØ
7Ø su=su+w:print#2,chr$(w);:next
8Ø w$=mid$(z$,65,2):gosub2ØØ:ps=w
9Ø w$=right$(z$,2):gosub2ØØ:ps=w*256+ps
11Ø z=z+1
12Ø print z
13Ø if z=1Ø77 then close2:close1:end
14Ø goto4Ø
2ØØ w=(asc(w$)-48+7*(asc(w$)>57))*16
21Ø w$=right$(w$,1)
22Ø w=w+asc(w$)-48+7*(w$>"9")
23Ø return
24Ø :
1ØØØ dataØcØ8ØaØØ9e2Ø3233Ø34ØØØØØØØØØØa9Ø18dØfcØa9ff8dff3f2Ø8Ø4c2Ø16Ø92Øe54c
1ØØ1 data4c2ØØ8ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1ØØ2 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1ØØ3 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1ØØ4 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1ØØ5 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1ØØ6 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1ØØ7 dataffffffffffffffffffffffffffffffffffa2ØØa9ØØ9dfcØb9dfcØc9dfcØd9dfcØee8
1ØØ8 datadØf14c894da9Ø285ffaØ3ea2Ø7989df8Ø788ca1Øf8a2Øfbd57Øa9dØØdØca1Øf7a9eØ
1ØØ9 data8d1ØdØe88a9dcØØa9dØØØe9dØØØfe8dØf4a9Øc9d27dØe8eØØ8dØf8a9ff8d15dØ8d1d
1Ø1Ø datadØa9ØØ8d17dØa2ØØaØ1286ac84ad855578aØ81a2Ø98c14Ø38e15Ø3a9ØØ8dØedca9f1
1Ø11 data8d1adØa9fa8d12dØ586Øa2Ø28e11dØad12dØdØfba23a8e11dØa9Ø18d19dØa9f18d1a
```

Listing 5/4.1.4-1 (Teil 1)

```
1012 datad020a0094c1fc0a000a23c183ec20f3ec10f3ec00f3e820f3e810f3e800f3e420f3e
1013 data410f3e400f3e020f3e010f3e000f3ec20e3ec10e3ec00e3e820e3e810e3e800e3e42
1014 data0e3e410e3e400e3e020e3e010e3e000e3ec20d3ec10d3ec00dcacacac8c015d0a6e6
1015 data55a555c908f00160a9008555a000b1acc900f012180a0a0a8554b015203c0ae6acd0
1016 data02e6ad60a90085aca91285ad4c0d0aa210e88e4a0a20440a4c1e0aa9108d4a0a4c44
1017 data0aa654a000bd001099c00fe8c8c8c8c018d0f26018ff48ff78ffa8ffd8ff08ff38ff
1018 data68ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1019 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1020 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1021 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1022 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1023 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1024 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1025 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1026 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1027 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1028 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1029 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1030 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1031 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1032 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1033 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1034 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1035 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1036 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1037 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1038 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1039 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1040 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1041 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1042 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1043 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1044 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1045 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1046 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1047 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1048 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1049 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1050 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1051 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1052 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1053 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1054 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1055 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1056 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1057 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1058 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1059 dataffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
1060 dataffffffffff3e73777770713e001c3e737f737373007e73737e73737e003e7370
1061 data7070733e007c76737373767c007f70707c70707f007f70707c707070003e73707773
1062 data733e0073737f737373003e1c1c1c1c1c3e001f0e0e0e0e763c0073767c787c7673
1063 data007070707070707f0063777f6b63636300737b7f7f777373003e73737373733e007e
1064 data73737e707070003e737373733e0f007e73737e7c7673003e73703e07733e007f1c1c
1065 data1c1c1c1c0073737373737333e0073737373733e1c006363636b7f77630073733e1c3e
1066 data7373007373733e1c1c1c007f070e1c38707f003e38383838383e000e19387e3871fe
1067 data003e0e0e0e0e0e3e00001c3e7f1c1c1c1c0018387f7f38180000000000000000001c
1068 data1c1c1c00001c00737373000000000007373ff73ff7373001c3f703e077e1c0071730e
1069 data1c387363003e733e3c67733f00070e1c000000000000e1c3838381c0e00381c0e0e0e
1070 data1c380000733eff3e730000001c1c7f1c1c00000000000000001c1c380000007f000000
1071 data0000000000001c1c000003070e1c3870003e73777b73733e001c1c3c1c1c1c7f003e
1072 data73070e38707f003e73071e07733e00070f1f737f0707007f707e0707733e003e7370
1073 data7e73733e007f730e1c1c1c1c003e73733e73733e003e73733f07733e0000001c0000
1074 data1c000000001c00001c1c380f1c3870381c0f000007f007f00000781c0e070e1c78
1075 data003e73070e1c001c000000000000000000000000000000000000000000000000
1076 data0000000000000000000000000000000000000000000000000000000000000000
```

Listing 5/4.1.4-1 (Teil 2)

```
10 rem demo-maker data lader2
20 :
30 z=1000:print"zeile:",z:open1,8,15,"i0":open2,8,2,"daten2.exe,p,w"
35 print#2,chr$(128);chr$(76);
40 read z$
50 su=0:for i=1to67 step2
60 w$=mid$(z$,i,2):gosub200
70 su=su+w:print#2,chr$(w);:next
80 w$=mid$(z$,65,2):gosub200:ps=w
90 w$=right$(z$,2):gosub200:ps=w*256+ps
110 z=z+1
120 print z
130 if z=1088 then close2:close1:end
140 goto40
200 w=(asc(w$)-48+7*(asc(w$)>57))*16
210 w$=right$(w$,1)
220 w=w+asc(w$)-48+7*(w$>"9")
230 return
240 :
1000 dataa93b8d11d0a9008d20d0a9008d21d0a9188d16d0a91d8d18d0a200bd403f9d0004bd
1001 data40409d0005bd40419d0006bd28429de806bd28439d00d8bd28449d00d9bd28459d00
1002 datadabd28469d00dbe8d0cda9018d0fc0a9ff8dff3f6020804c20160920e54c4cdf4cad
1003 data714da2009d27d0e8e008d0f8eee64cade64cc989d005a9718de64c4c044dee01d0ee
1004 data03d0ee05d0ee07d0ee09d0ee0bd0ee0dd0ee0fd0ad01d0c980f0034c354d78a9408d
1005 data024da94d8d034d584c354da208a0ff88d0fdcad0f860ce01d0ce03d0ce05d0ce07d0
1006 datace09d0ce0bd0ce0dd0ce0fd0ad01d0c950f0034c394d78a9048d024da94d8d034d58
1007 data4c394d060606060e0e0e0e0303030301010101030303030e0e0e0ea200bd28579d00
1008 data0ce8e080d0f5a200a9009d0020e8d0f8ee9c4dad9c4dc948d0eea9208d9c4da200bd
1009 data00109d0090bd00119d0091e8d0f1a94c8d1fc0a9318d20c0a9ea8d21c0201be5a90c
1010 data8d8602a201a00c200ce5a91aa053201eaba9178d18d0a9008d20d08d21d020e4ffc9
1011 data31f01fc932f01ec933f01dc934f01cc935f01bc936f01ac937f019c938f0184cf24d
1012 data4c304e4c884e4caa504c12514c6e514cc2514c96524c1753201be5a9e9a053201eab
1013 dataa90085c6a5c6c901d0fa201be5a90185cca9158d18d020e4fff0fbc985f00620d2ff
1014 data4c504ea200bd00049d0012bd00059d0013bd00069d0014bd00079d0015e8d0e5a901
1015 data85cca9178d18d04ccf4d201be5a9c0a054201eab20e4ffc931f00fc932f00ec933f0
1016 data0dc934f00c4c924e4c994f4cb44e4c214f4ccf4d201be5a99ea056201eaba00020cf
1017 dataff992e58c88402c011f007c90dd0ef4ce84ea9aea056201eaba90085c6a5c6c901d0
1018 datafa4cb44ea900992e58a9008d8602a901a208a00020baffa502a22ea05820bdffa900
1019 dataa200a09020d5ffa200bd00909d0010bd00919d0011e8d0f14ccf4d201be5a92fa056
1020 data201eab20e4ffc931f017c932f023c933f02fc934f03bc935f047c936f0104c2b4fa2
1021 data00bda8579d714de8e018d0f54ccf4da200bdc0579d714de8e018d0f54ccf4da200bd
1022 datad8579d714de8e018d0f54ccf4da200bdf0579d714de8e018d0f54ccf4da200bd0858
1023 data9d714de8e018d0f54ccf4d78a9938d1403a9508d1503a9008d0edca9f18d1ad0a9fa
1024 data8d12d058201be5a931a055201eaba90085c6a5c6c901d0faa9038d15d0a9308df807
1025 dataa9318df907a9a08d00d08d02d0ad5c4d8d01d0ad204d8d03d0a9018d27d08d28d020
1026 data1be5a912a056201eabad00dcc97ef024c97df029c96ff02e20e4ffc991f015c911f0
1027 data1ac90df01fa204a0ff88d0fdcad0f84cfd4fce5c4dce01d04c1b50ee5c4dee01d04c
1028 data1b50a2ffa0ff88d0fdcad0f8201be5a920a056201eabad00dcc97ef024c97df029c9
1029 data6ff02e20e4ffc991f015c911f01ac90df01fa204a0ff88d0fdcad0f84c4e50ce204d
1030 datace03d04c6c50ee204dee03d04c6c50a9008d15d04ccf4da2028e11d0ad12d0d0fba9
1031 data1a8d11d0a9018d19d04c31ea201be5a9d6a056201eaba000b92058992e58c8c007d0
1032 dataf520cfff992e58c88402c011f007c90dd0ef4ce950a9aea056201eaba90085c6a5c6
1033 datac901d0fa4caa50a92a88992e58a9008d8602a901a208a00020baffa502a22ea05820
1034 databdffa900a200a02020d5ff4ccf4d201be5a9caa056201eaba00020cfff992e58c884
1035 data02c011f007c90dd0ef4c4651a9aea056201eaba90085c6a5c6c901d0fa4c1251a900
1036 data992e58a9008d8602a901a208a00020baffa502a22ea05820bdffa900a200a0a020d5
1037 dataff4ccf4da9018d0fc0a9ff8dff3f20804c201609a90085c620e54ca5c6c901d0f720
1038 data18e5a9008d18d478a9318d1403a9ea8d1503a90085c6a90185cca91b8d11d0a9528d
1039 data12d0a92f85ffa9018d0edca9f08d1ad0584ccf4d201be5a9e3a056201eaba90085c6
1040 dataa5c6c901d0faa9308d0808a9368d0908a9348d0a08a900a2009d00489d00499d004a
1041 data9d004b9d004e9d004f9d0050e8d0e8ad04dc290f09308d2d58a900859d20e7ffa208
1042 dataa00120baffa904a228a05820bdffa901855fa9088560a95fa2ffa0cf86ae84afaab5
```

Listing 5/4.1.4-2 (Teil 1)

```
1043 data0085c1b50185c2a96185b920d5f3a5ba200ceda5b920b9eda000208efba5ac20dded
1044 dataa5ad20dded20d1fcb01478a2308601b1aca23686015820dded20dbfcd0e720feed24
1045 datab93011a5ba200ceda5b929ef09e020b9ed20feed18a937850120e2fc201be5a902a2
1046 data08a00020baffa227a057a90120bdff20c0ffa20220c6ff20cfffa590d00e20cfffa5
1047 datac6f023ad7702c903d01520ccffa90220c3ffa90d20d2ff20e4fff0fb4ccf4da90085
1048 datac68d770220cfff20cffff0dca90d20d2ffa91d20d2ff20d2ff20cfffaa20cfff20cd
1049 databda92020d2ff20cffff0b0a490d0b720d2ff4c08534ce2fcc4454d4f20cd414b4552
1050 data20d6312e310d0d20202020202020202028432920c82e20d34348524f455445522038
1051 data2f38390d0d0d2020312dcc41554653434852494654544558542045494e474542454e
1052 data0d0d2020322dcc415546534348524946542 0cd454e55450d0d2020332dc752414649
1053 data4b204c4144454e0d0d2020342dcd5553494b204c4144454e0d0d2020352dc4454d4f
1054 data205a454947454e0d0d2020362dc4454d4f2053504549434845524e0d0d2020372dc4
1055 data49524543544f5259205a454947454e0d0d2020382dc54e44450d000dcd4954204449
1056 data4553454d20c54449544f52204b4f454e4e454e20d349452044454e0dd44558542c20
1057 data444552204e41434848455220494e20494852454d20c4454d4f20414c530dcc415546
1058 data53434P5249465420455253434845494e454e20534f4c4c2c2053434852454942454e
1059 data2e0dc14d20c54e44452044455320d44558544553204d55455353454e20d349452045
1060 data494e20274027200d45494e474542454e2e20d645524c415353454e2044455320c544
1061 data49544f52532044555243480d44494520d4415354452027c6312720210d0d0d20d4c1
1062 datad3d4c5000dcc4155465343485249465420cd454e55450d0d2020312dd35052494e47
1063 data424552454943482046455354 4c4547454e0d0d2020322dda454943485341545a204c
1064 data4144454e0d0d2020332dc6415242454e424c494e4b454e2041454e4445524e0d0d20
1065 data20342dc84155505420cd454e5545000dd54d2044454e20d35052494e474245524549
1066 data434820464553545a554c4547454e2c0d4d55455353454e20d34945204d495420c849
1067 data4c46452044455320ca4f59535449434b530d494e20d04f525432204f444552204445
1068 data4e20c3d2d3d220d4415354454e204449450d4f4245524520425a572e204449452055
1069 data4e5445524520c752454e5a4520444546494e494552454e2e0dda554d20c64553544c
1070 data4547454e2044525545434b454e20d349452042495454452044494 50dd2c5d4d5d2ce
1071 data2dd441535445204f4445522044454e20c6455545524b4e4f50462e0d0d0dd4415354
1072 data45000dcf4245524520c752454e5a45000dd54e5445524520c752454e5a45000dc94e
1073 data2057454c4348454e20c641524254 4f454e454e20534f4c4c20444945 20cc4155462d
1074 data0d53434852494654204 24c494e4b454e203f0d0d0d312dc24c41550d0d322dc75255
1075 data454e0d0d332dd24f540d0d342dc75241550d0d352dc7454c420d0d362dc841555054
1076 data4d454e5545000dda454943485341545a4e414d453a000dce414d4520495354205a55
1077 data204c414e47210d0d0d0dd441535445000dcd5553494b4e414d453a000dc752414649
1078 data4b4e414d453a000dc4415320c4454d4f20574952442041554620d4415354454e4452
1079 data55434b0d0d41554620c449534b4554544520474553504549434845 52542e200d0d0d
1080 data0dd44153544500240000000000000000000000000000000000000000000000000000
1081 data00000000ffffff001000003800007c0000fe000010000010000010000000000000000
1082 data0000000000000000000000000000100000100000100000fe00007c00003800001000
1083 dataffffff00000000000000000000000000000000000000000000000000000000000000
1084 data060606060e0e0e0e03030303010101010303030 30e0e0e0e05050505030303030d0d
1085 data0d0d010101010d0d0d0d030303030404040402020202 0a0a0a0a010101010a0a0a0a
1086 data020202020b0b0b0b0c0c0c0c0f0f0f0f010101010f0f0f0f0c0c0c0c090909090808
1087 data08080707070701010101070707070808080 83f504943203f200044454d4f00000000
```

Listing 5/4.1.4-2 (Teil 2)

```
                .ba $0801
                .ou "@0:demo maker"
                .wa

sprites:        .eq $07f8
sp:             .eq $d000
sirq:           .eq $ea31
sirq2:          .eq $c01f
cls:            .eq $e51b
cursorf:        .eq $0286
strout:         .eq $ab1e
cursor:         .eq $e50c
getin:          .eq $ffe4
bsout:          .eq $ffd2
cls2:           .eq $e518
reset:          .eq $fce2
fpas:           .eq $ffba
fnpas:          .eq $ffbd
open:           .eq $ffc0
chkin:          .eq $ffc6
basin:          .eq $ffcf
lhb:            .eq $bdcd
clrch:          .eq $ffcc
close:          .eq $ffc3
load:           .eq $ffd5
clall:          .eq $ffe7
s1:             .eq $f3d5
s2:             .eq $ed0c
s3:             .eq $edb9
s4:             .eq $fb8e
s5:             .eq $eddd
s6:             .eq $fcd1
s7:             .eq $fcdb
s8:             .eq $edfe

basic:          .by $0c,$08,$0a,$00,$9e,$20,$32,$33,$30,$34,0,0,0,0,0

dsta:           lda #$01
                sta $c00f           ; Music Neustart
                lda #$ff
                sta $3fff           ; Randmuster voll
                jsr spic            ; Bild zeigen
                jsr scroll          ; Laufschrift starten
dsta1:          jsr cc1             ; Spriteblinken
                jmp dsta1

bfree:          .db $da

                ldx #$00
                lda #$00
loopa:          sta mem1+$195,x
                sta mem1+$295,x
                sta mem1+$395,x
                sta mem1+$495,x
                inx
                bne loopa           ; Scrollbereich loeschen

                jmp start1          ; Editor starten

scroll:         lda #$02
                sta $ff
                ldy #$3e
```

Listing 5/4.1.4-3 (Teil 1)

```
        ldx #$07
xx1:    tya
        sta sprites,x       ; Spriteblocks definieren
        dey
        dex
        bpl xx1
        ldx #$0f
xx2:    lda spos1,x         ; Spritepositionen festlegen
        sta sp,x
        dex
        bpl xx2
        lda #$e0
        sta sp+$10
        inx
        txa
xx3:    sta mem1+$59,x
        sta mem1+$399,x
        sta mem1+$499,x
        inx
        bne xx3
        lda #$0c
xx4:    sta sp+$27,x
        inx
        cpx #08
        bne xx4
        lda #$ff
        sta sp+$15          ; Sprites einschalten
        sta sp+$1d          ; S. in X Richtung vergroessern
        lda #00
        sta sp+$17          ; S. in Y Richtung normal
        ldx #<stext
        ldy #>stext
        stx $ac
        sty $ad             ; Scrolltext Adressen in ac +ad
        sta $55
        sei
        ldy #<irq
        ldx #>irq
        sty $0314
        stx $0315           ; Interrupt setzen
        lda #00
        sta $dc0e
        lda #$f1
        sta $d01a
        lda #$fa
        sta $d012
        cli
        rts
irq:    ldx #$02
        stx $d011
xx5:    lda $d012
        bne xx5
        ldx #$3a
        stx $d011
        lda #01
        sta $d019
        lda #$f1
        sta $d01a
        jsr mtext
        jmp sirq2

mtext:  ldy #00
        ldx #$3c
```

Listing 5/4.1.4-3 (Teil 2)

```
xx6:        clc
            rol mem1+$55b,x
            rol mem1+$55a,x
            rol mem1+$559,x
            rol mem1+$51b,x
            rol mem1+$51a,x
            rol mem1+$519,x
            rol mem1+$4db,x
            rol mem1+$4da,x
            rol mem1+$4d9,x
            rol mem1+$49b,x
            rol mem1+$49a,x
            rol mem1+$499,x
            rol mem1+$45b,x
            rol mem1+$45a,x
            rol mem1+$459,x
            rol mem1+$41b,x
            rol mem1+$41a,x
            rol mem1+$419,x
            rol mem1+$3db,x
            rol mem1+$3da,x
            rol mem1+$3d9,x
            rol mem1+$39b,x
            rol mem1+$39a,x
            rol mem1+$399,x
            rol mem1+$35b,x
            rol mem1+$35a,x
            rol mem1+$359,x
            dex
            dex
            dex
            iny
            cpy #$15
            bne xx6              ; Text scrollen
            inc $55
            lda $55
            cmp #$08
            beq xx7
            rts

xx7:        lda #00
            sta $55
            ldy #00
xx12:       lda ($ac),y
            cmp #00
            beq xx9
            clc
            asl
            asl
            asl
            sta $54
            bcs xx10
            jsr xx11
xx13:       inc $ac
            bne xx8
            inc $ad
xx8:        rts

xx9:        lda #<stext
            sta $ac
            lda #>stext
            sta $ad              ; Textadressen in ac + ad
            jmp xx12
```

Listing 5/4.1.4-3 (Teil 3)

```
xx10:       ldx #>zei          ; Zeichensatzadresse +1 zum Auslesen
            inx                ; festlegen
            stx xa+6
            jsr xa
            jmp xx13
xx11:       lda #>zei          ; Zeichensatzadresse festlegen
            sta xa+6
            jmp xa

xa:         ldx $54
            ldy #00
xa1:        lda zei,x          ; Zeichensatz auslesen
            sta mem1+$559,y
            inx
            iny
            iny
            iny
            cpy #$18
            bne xa1
            rts

spos1:      .by $18,$ff,$48,$ff,$78,$ff,$a8,$ff
            .by $d8,$ff,$08,$ff,$38,$ff,$68,$ff

mem1:
hh1:        .db $599

zei:        .by $3e,$73,$77,$77,$70,$71,$3e,$00
            .by $1c,$3e,$73,$7f,$73,$73,$73,$00
            .by $7e,$73,$73,$7e,$73,$73,$7e,$00
            .by $3e,$73,$70,$70,$70,$73,$3e,$00
            .by $7c,$76,$73,$73,$73,$76,$7c,$00
            .by $7f,$70,$70,$7c,$70,$70,$7f,$00
            .by $7f,$70,$70,$7c,$70,$70,$70,$00
            .by $3e,$73,$70,$77,$73,$73,$3e,$00
            .by $73,$73,$73,$7f,$73,$73,$73,$00
            .by $3e,$1c,$1c,$1c,$1c,$1c,$3e,$00
            .by $1f,$0e,$0e,$0e,$0e,$76,$3c,$00
            .by $73,$76,$7c,$78,$7c,$76,$73,$00
            .by $70,$70,$70,$70,$70,$70,$7f,$00
            .by $63,$77,$7f,$6b,$63,$63,$63,$00
            .by $73,$7b,$7f,$7f,$77,$73,$73,$00
            .by $3e,$73,$73,$73,$73,$73,$3e,$00
            .by $7e,$73,$73,$7e,$70,$70,$70,$00
            .by $3e,$73,$73,$73,$73,$3e,$0f,$00
            .by $7e,$73,$73,$7e,$7c,$76,$73,$00
            .by $3e,$73,$70,$3e,$07,$73,$3e,$00
            .by $7f,$1c,$1c,$1c,$1c,$1c,$1c,$00
            .by $73,$73,$73,$73,$73,$73,$3e,$00
            .by $73,$73,$73,$73,$73,$3e,$1c,$00
            .by $63,$63,$63,$6b,$7f,$77,$63,$00
            .by $73,$73,$3e,$1c,$3e,$73,$73,$00
            .by $73,$73,$73,$3e,$1c,$1c,$1c,$00
            .by $7f,$07,$0e,$1c,$38,$70,$7f,$00
            .by $3e,$38,$38,$38,$38,$38,$3e,$00
            .by $0e,$19,$38,$7e,$38,$71,$fe,$00
            .by $3e,$0e,$0e,$0e,$0e,$0e,$3e,$00
            .by $00,$1c,$3e,$7f,$1c,$1c,$1c,$1c
            .by $00,$18,$38,$7f,$7f,$38,$18,$00
            .by $00,$00,$00,$00,$00,$00,$00,$00
            .by $1c,$1c,$1c,$1c,$00,$00,$1c,$00
```

Listing 5/4.1.4-3 (Teil 4)

```
                .by $73,$73,$73,$00,$00,$00,$00,$00
                .by $73,$73,$ff,$73,$ff,$73,$73,$00
                .by $1c,$3f,$70,$3e,$07,$7e,$1c,$00
                .by $71,$73,$0e,$1c,$38,$73,$63,$00
                .by $3e,$73,$3e,$3c,$67,$73,$3f,$00
                .by $07,$0e,$1c,$00,$00,$00,$00,$00
                .by $0e,$1c,$38,$38,$38,$1c,$0e,$00
                .by $38,$1c,$0e,$0e,$0e,$1c,$38,$00
                .by $00,$73,$3e,$ff,$3e,$73,$00,$00
                .by $00,$1c,$1c,$7f,$1c,$1c,$00,$00
                .by $00,$00,$00,$00,$00,$1c,$1c,$38
                .by $00,$00,$00,$7f,$00,$00,$00,$00
                .by $00,$00,$00,$00,$00,$1c,$1c,$00
                .by $00,$03,$07,$0e,$1c,$38,$70,$00
                .by $3e,$73,$77,$7b,$73,$73,$3e,$00
                .by $1c,$1c,$3c,$1c,$1c,$1c,$7f,$00
                .by $3e,$73,$07,$0e,$38,$70,$7f,$00
                .by $3e,$73,$07,$1e,$07,$73,$3e,$00
                .by $07,$0f,$1f,$73,$7f,$07,$07,$00
                .by $7f,$70,$7e,$07,$07,$73,$3e,$00
                .by $3e,$73,$70,$7e,$73,$73,$3e,$00
                .by $7f,$73,$0e,$1c,$1c,$1c,$1c,$00
                .by $3e,$73,$73,$3e,$73,$73,$3e,$00
                .by $3e,$73,$73,$3f,$07,$73,$3e,$00
                .by $00,$00,$1c,$00,$00,$1c,$00,$00
                .by $00,$00,$1c,$00,$00,$1c,$1c,$38
                .by $0f,$1c,$38,$70,$38,$1c,$0f,$00
                .by $00,$00,$7f,$00,$7f,$00,$00,$00
                .by $78,$1c,$0e,$07,$0e,$1c,$78,$00
                .by $3e,$73,$07,$0e,$1c,$00,$1c,$00

stext:
hh:             .db $3a80
spic:           lda #$3b
                sta $d011
                lda #00
                sta $d020
                lda #00
                sta $d021          ; Bildschirmfarben schwarz
                lda #$18
                sta $d016
                lda #$1d
                sta $d018          ; Hires einschalten
                ldx #$00
y1:             lda $3f40,x
                sta $0400,x
                lda $4040,x
                sta $0500,x
                lda $4140,x
                sta $0600,x
                lda $4228,x
                sta $06e8,x
                lda $4328,x
                sta $d800,x
                lda $4428,x
                sta $d900,x
                lda $4528,x
                sta $da00,x
                lda $4628,x
                sta $db00,x
                inx
```

Listing 5/4.1.4-3 (Teil 5)

```
                bne y1              ; Bilddaten auf Bildschirm
                lda #01
                sta $c00f
                lda #$ff
                sta $3fff
                rts
stde:           jsr spic
                jsr scroll
y2:             jsr cc1
                jmp y2
cc1:            lda ccycle
                ldx #$00
cc2:            sta sp+$27,x        ; Spritefarben setzen
                inx
                cpx #$08
                bne cc2
                inc cc1+1
                lda cc1+1
                cmp #<ccycle+24
                bne cc3
                lda #<ccycle
                sta cc1+1
cc3:            jmp cc4

cc4:            inc $d001
                inc $d003
                inc $d005
                inc $d007
                inc $d009
                inc $d00b
                inc $d00d
                inc $d00f
                lda $d001
cc4a:           cmp #$80            ; S.Positionen runter
                beq cc5
                jmp cc6

cc5:            sei
                lda #<cc7
                sta cc3+1
                lda #>cc7
                sta cc3+2
                cli
                jmp cc6

cc6:            ldx #$08
cc9:            ldy #$ff
cc8:            dey
                bne cc8
                dex
                bne cc9             ; Verzoegerungsschleife
                rts

cc7:            dec $d001
                dec $d003
                dec $d005
                dec $d007
                dec $d009
                dec $d00b
                dec $d00d
                dec $d00f
                lda $d001
```

Listing 5/4.1.4-3 (Teil 6)

```
cc7a:       cmp #$50            ; S.Positionen rauf
            beq cc10
            jmp cc8

cc10:       sei
            lda #<cc4
            sta cc3+1
            lda #>cc4
            sta cc3+2
            cli
            jmp cc8
ccycle:     .by 6,6,6,6,14,14,14,14,3,3,3,3,1,1,1,1,3,3,3,3,14,14,14,14


start1:     ldx #00
c0:         lda sprite,x
            sta $0c00,x
            inx
            cpx #$80
            bne c0              ; Sprites definieren
            ldx #$00
c1:         lda #$00
            sta $2000,x
            inx
            bne c1              ; Bitmap loeschen
            inc c1+4
            lda c1+4
            cmp #$48
            bne c1
            lda #$20
            sta c1+4
            ldx #$00
c2:         lda zei,x
            sta $9000,x
            lda zei+$100,x
            sta $9100,x
            inx
            bne c2              ; Zeichensatz nach $9000 kopieren
            lda #$4c
            sta $c01f
            lda #$31
            sta $c020
            lda #$ea
            sta $c021
start:      jsr cls             ; Bildschirm loeschen
            lda #12
            sta cursorf         ; Cursorfarbe setzen
            ldx #01             ; Zeile
            ldy #12             ; Spalte
            jsr cursor          ; Cursor setzen
            lda #<text
            ldy #>text
            jsr strout
            lda #$17
            sta $d018
            lda #00
            sta $d020
            sta $d021           ; Bildschirmfarben = schwarz

loop1:      jsr getin           ; hole Tastatureingabe
            cmp #$31            ; vergleiche mit 1
            beq eins            ; wenn ja, springe zu eins
```

Listing 5/4.1.4-3 (Teil 7)

```
            cmp #$32          ; vergleiche mit 2
            beq zwei          ; wenn ja, springe zu zwei
            cmp #$33          ; vergleiche mit 3
            beq drei          ; wenn ja, springe zu drei
            cmp #$34          ; vergleiche mit 4
            beq vier          ; wenn ja, springe zu vier
            cmp #$35          ; vergleiche mit 5
            beq fuenf         ; wenn ja, springe zu fuenf
            cmp #$36          ; vergleiche mit 6
            beq sechs         ; wenn ja, springe zu sechs
            cmp #$37          ; vergleiche mit 7
            beq sieben        ; wenn ja, springe zu sieben
            cmp #$38          ; vergleiche mit 8
            beq acht          ; wenn ja, springe zu acht
            jmp loop1

eins:       jmp einsa         ; springe zu einsa
zwei:       jmp zweia         ; springe zu zweia
drei:       jmp dreia         ; springe zu dreia
vier:       jmp viera         ; springe zu viera
fuenf:      jmp fuenfa        ; springe zu fuenfa
sechs:      jmp sechsa        ; springe zu sechsa
sieben:     jmp siebena       ; springe zu siebena
acht:       jmp achta         ; springe zu achta

einsa:      jsr cls           ; loesche Bildschirm
            lda #<text2
            ldy #>text2
            jsr strout
            lda #00
            sta $c6
loop2:      lda $c6
            cmp #01
            bne loop2         ; auf Tastendruck warten
            jsr cls           ; Bildschirm loeschen
            lda #01
            sta $cc           ; Cursor aus
            lda #$15
            sta $d018         ; Zeichsatz Gross/Grafik
loop3:      jsr getin
            beq loop3
            cmp #$85          ; vergleiche Tastendruck mit F1
            beq tstop         ; wenn F1, springe zu tstop
            jsr bsout         ; schreibe Zeichen auf Bildschirm
            jmp loop3

tstop:      ldx #$00
tstop1:     lda $0400,x
            sta stext,x
            lda $0500,x
            sta stext+$100,x
            lda $0600,x
            sta stext+$200,x
            lda $0700,x
            sta stext+$300,x
            inx
            bne tstop1        ; Scrolltext sichern
            lda #01
            sta $cc
            lda #$17
            sta $d018         ; Zeichensatz Gross/Klein
            jmp start
```

Listing 5/4.1.4-3 (Teil 8)

```
zweia:      jsr cls            ; Bildschirm loeschen
            lda #<text3
            ldy #>text3
            jsr strout
zweib:      jsr getin
            cmp #$31
            beq zweic
            cmp #$32
            beq zweid
            cmp #$33
            beq zweie
            cmp #$34
            beq zweif          ; Tastendruck auswerten
            jmp zweib
zweic:      jmp zweics
zweid:      jmp zweids
zweie:      jmp zweies
zweif:      jmp start

zweids:     jsr cls
            lda #<text8
            ldy #>text8
            jsr strout
            ldy #$00
zweid2:     jsr basin
            sta buffer,y       ; Zeichensatzname holen + speicherern
            iny
            sty $02
            cpy #$11
            beq zweid1
            cmp #$0d
            bne zweid2
            jmp zweid3
zweid1:     lda #<text9
            ldy #>text9
            jsr strout
            lda #$00
            sta $c6
zweid4:     lda $c6
            cmp #$01
            bne zweid4         ; auf Tastendruck warten
            jmp zweids
zweid3:     lda #$00
            sta buffer,y
            lda #$00
            sta cursorf        ; Cursorfarbe schwarz
            lda #$01
            ldx #$08
            ldy #$00
            jsr fpas           ; File Parameter setzen
            lda $02
            ldx #<buffer
            ldy #>buffer
            jsr fnpas          ; Filename Parameter setzen
            lda #$00
            ldx #$00
            ldy #$90
            jsr load           ; File laden
            ldx #$00
zweid5:     lda $9000,x
            sta zei,x
            lda $9100,x
```

Listing 5/4.1.4-3 (Teil 9)

```
            sta zei+$100,x
            inx
            bne zweid5          ; ZS kopieren
            jmp start

zweies:     jsr cls
            lda #<text7
            ldy #>text7
            jsr strout
zweie1:     jsr getin
            cmp #$31
            beq zweie2
            cmp #$32
            beq zweie3
            cmp #$33
            beq zweie4
            cmp #$34
            beq zweie5
            cmp #$35
            beq zweie6
            cmp #$36
            beq zweie7
            jmp zweie1          ; Farben auswaehlen
zweie2:     ldx #$00
            lda farbe1,x
            sta ccycle,x
            inx
            cpx #$18
            bne zweie2+2
zweie7:     jmp start
zweie3:     ldx #$00
            lda farbe2,x
            sta ccycle,x
            inx
            cpx #$18
            bne zweie3+2
            jmp start
zweie4:     ldx #$00
            lda farbe3,x
            sta ccycle,x
            inx
            cpx #$18
            bne zweie4+2
            jmp start
zweie5:     ldx #$00
            lda farbe4,x
            sta ccycle,x
            inx
            cpx #$18
            bne zweie5+2
            jmp start
zweie6:     ldx #$00
            lda farbe5,x
            sta ccycle,x
            inx
            cpx #$18
            bne zweie6+2
            jmp start           ; Farben kopieren

zweics:     sei
            lda #<zweic1
            sta $0314
```

Listing 5/4.1.4-3 (Teil 10)

```
            lda #>zweic1
            sta $0315
            lda #00
            sta $dc0e
            lda #$f1
            sta $d01a
            lda #$fa
            sta $d012
            cli                  ; Rand ausschalten
            jsr cls
            lda #<text4
            ldy #>text4
            jsr strout
            lda #$00
            sta $c6
zweic0:     lda $c6
            cmp #$01
            bne zweic0
            lda #$03
            sta sp+$15
            lda #$30
            sta sprites
            lda #$31
            sta sprites+1        ;Sprites setzen
            lda #$a0
            sta sp
            sta sp+2
            lda cc7a+1
            sta sp+1
            lda cc4a+1
            sta sp+3
            lda #$01
            sta sp+$27
            sta sp+$28
            jsr cls
            lda #<text5
            ldy #>text5
            jsr strout
zweic3:     lda $dc00
            cmp #$7e
            beq zweic6
            cmp #$7d
            beq zweic7
            cmp #$6f
            beq zweic8           ; Joystick auswerten
            jsr getin
            cmp #$91
            beq zweic6
            cmp #$11
            beq zweic7
            cmp #$0d
            beq zweic8           ; Tastatur auswerten
zweic9:     ldx #$04
zweic5:     ldy #$ff
zweic4:     dey
            bne zweic4
            dex
            bne zweic5
            jmp zweic3
zweic6:     dec cc7a+1
            dec sp+1
            jmp zweic9
```

Listing 5/4.1.4-3 (Teil 11)

```
zweic7:     inc cc7a+1
            inc sp+1
            jmp zweic9
zweic8:     ldx #$ff
zweic10:    ldy #$ff
zweic11:    dey
            bne zweic11
            dex
            bne zweic10
            jsr cls
zweic12:    lda #<text6
            ldy #>text6
            jsr strout
zweic15:    lda $dc00
            cmp #$7e
            beq zweic16
            cmp #$7d
            beq zweic17
            cmp #$6f
            beq zweic18          ; J. auswerten
            jsr getin
            cmp #$91
            beq zweic16
            cmp #$11
            beq zweic17
            cmp #$0d
            beq zweic18          ; T. auswerten
zweic19:    ldx #$04
zweic14:    ldy #$ff
zweic13:    dey
            bne zweic13
            dex
            bne zweic14
            jmp zweic15
zweic16:    dec cc4a+1
            dec sp+3
            jmp zweic19
zweic17:    inc cc4a+1
            inc sp+3
            jmp zweic19
zweic18:    lda #$00
            sta sp+$15
            jmp start
zweic1:     ldx #$02
            stx $d011
zweic2:     lda $d012
            bne zweic2
            lda #$1a
            sta $d011
            lda #01
            sta $d019
            jmp sirq
dreia:      jsr cls
            lda #<text11
            ldy #>text11
            jsr strout
            ldy #$00
dreib:      lda bda,y
            sta buffer,y
            iny
            cpy #$07
            bne dreib
```

Listing 5/4.1.4-3 (Teil 12)

```
              lda #$00
              sta cursorf
              lda #$01
              ldx #$08
              ldy #$00
              jsr fpas
              lda $02
              ldx #<buffer
              ldy #>buffer
              jsr fnpas
              lda #$00
              ldx #$00
              ldy #$a0
              jsr load            ; Musik laden
              jmp start

fuenfa:       lda #01
              sta $c00f
              lda #$ff
              sta $3fff
              jsr spic
              jsr scroll          ; Demo zeigen
              lda #00
              sta $c6
fuenfb:       jsr ccl
              lda $c6
              cmp #01
              bne fuenfb
              jsr cls2
              lda #00
              sta $d418
              sei
              lda #$31
              sta $0314
              lda #$ea
              sta $0315           ; IRQ normal weiter
              lda #00
              sta $c6
              lda #01
              sta $cc
              lda #$1b
              sta $d011
              lda #$52
              sta $d012
              lda #$2f
              sta $ff
              lda #01
              sta $dc0e
              lda #$f0
              sta $d01a
              cli
              jmp start

sechsa:       jsr cls
              lda #<text12
              ldy #>text12
              jsr strout
              lda #$00
              sta $c6
sechsb:       lda $c6
              cmp #$01
              bne sechsb
```

Listing 5/4.1.4-3 (Teil 13)

```
dreic:      jsr basin
            sta buffer,y        ;Grafikname holen + speichern
            iny
            sty $02
            cpy #$11
            beq dreid
            cmp #$0d
            bne dreic
            jmp dreif
dreid:      lda #<text9
            ldy #>text9
            jsr strout
            lda #$00
            sta $c6
dreie:      lda $c6
            cmp #$01
            bne dreie
            jmp dreia
dreif:      lda #$2a
            dey
            sta buffer,y
            lda #$00
            sta cursorf
            lda #$01
            ldx #$08
            ldy #$00
            jsr fpas
            lda $02
            ldx #<buffer
            ldy #>buffer
            jsr fnpas
            lda #$00
            ldx #$00
            ldy #$20
            jsr load            ; Grafik laden
            jmp start

viera:      jsr cls
            lda #<text10
            ldy #>text10
            jsr strout
            ldy #$00
viera1:     jsr basin
            sta buffer,y        ; Musikname holen + speichern
            iny
            sty $02
            cpy #$11
            beq viera2
            cmp #$0d
            bne viera1
            jmp viera4
viera2:     lda #<text9
            ldy #>text9
            jsr strout
            lda #$00
            sta $c6
viera3:     lda $c6
            cmp #$01
            bne viera3
            jmp viera
viera4:     lda #$00
            sta buffer,y
```

Listing 5/4.1.4-3 (Teil 14)

```
            lda #$30
            sta basic+7
            lda #$36
            sta basic+8
            lda #$34
            sta basic+9
            lda #$00
            ldx #$00
del:        sta $4800,x
            sta $4900,x
            sta $4a00,x
            sta $4b00,x
            sta $4e00,x
            sta $4f00,x
            sta $5000,x
            inx
            bne del              ; Editor loeschen
            lda $dc04
            and #$0f
            ora #$30
            sta wert
            lda #$00
            sta $9d
            jsr clall
            ldx #$08
            ldy #$01
            jsr fpas
            lda #$04
            ldx #<dname
            ldy #>dname
            jsr fnpas
            lda #$01
            sta $5f
            lda #$08
            sta $60
            lda #$5f
            ldx #$ff
            ldy #$cf
            stx $ae
            sty $af
            tax
            lda $00,x
            sta $c1
            lda $01,x
            sta $c2
            lda #$61
            sta $b9
            jsr s1
            lda $ba
            jsr s2
            lda $b9
            jsr s3
            ldy #$00
            jsr s4
            lda $ac
            jsr s5
            lda $ad
            jsr s5
sechse:     jsr s6
            bcs sechse
            sei
            ldx #$30
```

Listing 5/4.1.4-3 (Teil 15)

```
            stx $01
            lda $(ac),y
            ldx #$36
            stx $01
            cli
            jsr s5
            jsr s7
            bne sechsc
sechse:     jsr s8
            bit $b9
            bmi sechsd
            lda $ba
            jsr s2
            lda $b9
            and #$ef
            ora #$e0
            jsr s3
            jsr s8            ; Demo speichern
sechsd:     clc
            lda #$37
            sta $01
            jsr reset         ; Reset

siebena:    jsr cls
            lda #$02
            ldx #$08
            ldy #$00
            jsr fpas
            ldx #<dir
            ldy #>dir
            lda #$01
            jsr fnpas
            jsr open
            ldx #$02
            jsr chkin
            jsr basin
            lda $90
            bne siebenb
            jsr basin
siebenf:    lda $c6
            beq siebenc
            lda $0277
            cmp #$03
            bne siebend
siebenb:    jsr clrch
            lda #$02
            jsr close
            lda #$0d
            jsr bsout
siebene:    jsr getin
            beq siebene
            jmp start
siebend:    lda #$00
            sta $c6
            sta $0277
siebenc:    jsr basin
            jsr basin
            beq siebenb
            lda #$0d
            jsr bsout
            lda #$1d
            jsr bsout
```

Listing 5/4.1.4-3 (Teil 16)

```
                jsr bsout
                jsr basin
                tax
                jsr basin
                jsr lhb
                lda #$2Ø
                jsr bsout
siebeng:        jsr basin
                beq siebenf
                ldy $8Ø
                bne siebenb
                jsr bsout
                jmp siebeng


achta:          jmp reset          ; bei F8 verlassen durch Reset


text:           .by "Demo Maker V1.1",13,13
                .by "         (c) H. Schroeter 8/88",13,13,13
                .by "  1-Laufschrifttext eingeben",13,13
                .by "  2-Laufschrift Menue",13,13
                .by "  3-Grafik laden",13,13
                .by "  4-Musik laden",13,13
                .by "  5-Demo zeigen",13,13
                .by "  6-Demo speichern",13,13
                .by "  7-Directory zeigen",13,13
                .by "  8-Ende",13
                .by Ø
text2:          .by 13,"Mit diesem Editor koennen Sie den",13
                .by "Text, der nachher in ihrem Demo als",13
                .by "Laufschrift erscheinen soll, schreiben.",13
                .by "Am Ende des Textes muessen Sie ein '@' ",13
                .by "eingeben. Verlassen des Editors durch",13
                .by "die Taste 'F1' !",13,13,13
                .by " TASTE"
                .by Ø

text3:          .by 13,"Laufschrift Menue",13,13
                .by "  1-Springbereich festlegen",13,13
                .by "  2-Zeichsatz laden",13,13
                .by "  3-Farbenblinken aendern",13,13
                .by "  4-Haupt Menue",Ø

text4:          .by 13,"Um den Springbereich festzulegen,",13
                .by "muessen Sie mit Hilfe des Joysticks",13
                .by "in Port2 oder den CRSR Tasten die",13
                .by "obere bzw. die untere Grenze definieren.",13
                .by "Zum Festlegen druecken Sie bitte die",13
                .by "RETURN-Taste oder den Feuerknopf.",13,13,13
                .by "Taste",Ø

text5:          .by 13,"Obere Grenze",Ø
text6:          .by 13,"Untere Grenze",Ø

text7:          .by 13,"In welchen Farbtoenen soll die Lauf-",13
                .by "schrift blinken ?",13,13,13
                .by "1-Blau",13,13
                .by "2-Gruen",13,13
                .by "3-Rot",13,13
```

Listing 5/4.1.4-3 (Teil 17)

```
               .by "4-Grau",13,13
               .by "5-Gelb",13,13
               .by "6-Hauptmenue",Ø

text8:         .by 13,"Zeichsatzname:",Ø
text9:         .by 13,"Name ist zu lang!",13,13,13
               .by 13,"Taste",Ø
text1Ø:        .by 13,"Musikname:",Ø
text11:        .by 13,"Grafikname:",Ø

text12:        .by 13,"Das Demo wird auf Tastendruck",13
               .by 13,"auf Diskette gespeichert. ",13,13,13
               .by 13,"Taste",Ø

dir:           .by "$"
sprite:        .by $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ
               .by $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ
               .by $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ
               .by $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ff,$ff
               .by $ff,$ØØ,$1Ø,$ØØ,$ØØ,$38,$ØØ,$ØØ
               .by $7c,$ØØ,$ØØ,$fe,$ØØ,$ØØ,$1Ø,$ØØ
               .by $ØØ,$1Ø,$ØØ,$ØØ,$1Ø,$ØØ,$ØØ,$ØØ
               .by $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ
               .by $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ
               .by $ØØ,$ØØ,$1Ø,$ØØ,$ØØ,$1Ø,$ØØ,$ØØ
               .by $1Ø,$ØØ,$ØØ,$fe,$ØØ,$ØØ,$7c,$ØØ
               .by $ØØ,$38,$ØØ,$ØØ,$1Ø,$ØØ,$ff,$ff
               .by $ff,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ
               .by $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ
               .by $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ
               .by $ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ,$ØØ
farbe1:        .by 6,6,6,6,14,14,14,14,3,3,3,3,1,1,1,1,3,3,3,3,14,14,14,14

farbe2:        .by 5,5,5,5,3,3,3,3,13,13,13,13,1,1,1,1,13,13,13,13,3,3,3,3

farbe3:        .by 4,4,4,4,2,2,2,2,1Ø,1Ø,1Ø,1Ø,1,1,1,1,1Ø,1Ø,1Ø,1Ø,2,2,2,2

farbe4:        .by 11,11,11,11,12,12,12,12,15,15,15,15,1,1,1,1,15,15,15,15
               .by 12,12,12,12

farbe5:        .by 9,9,9,9,8,8,8,8,7,7,7,7,1,1,1,1,7,7,7,7,8,8,8,8

bda:           .by $3f,$5Ø,$49,$43,$2Ø,$3f,$2Ø,Ø

dname:         .by "demo",Ø

wert:          .by Ø

buffer:
```

Listing 5/4.1.4-3 (Teil 18)

5/5

Spiele

5/5.2

ALPHA V

(Autor: Rainer Gebhardt)

Gute Nerven und ein kühler Kopf sind bei dem im folgenden abgedruckten Spiel mit Sicherheit sehr hilfreich, denn es wird schwieriger, als Sie sich denken.

Benötigt werden:
- C 64/C 128
- Joystick
- Nerven

5/5.2.1

Spielanleitung

Kurze Vorgeschichte: Das Flaggschiff der Erde, Delta 1, geriet in einen Asteroidensturm; dabei wurden die Treibstofftanks so schwer beschädigt, daß das Schiff Fähren zum Planeten ALPHA V schicken muß, um dort Treibstoff zu holen. Sie als Kommander des Schiffes müssen auf dem Planeten ALPHA V landen und heil wieder zum Schiff zurückfliegen. Nach zweimaligem Landen auf dem Planeten und im Schiff fliegt das Schiff weiter bis zum nächsten Bild.

Die Fähre wird durch den Joystick an Port 2 gelenkt.

Joystick: Drücken nach links	—	Fähre nach links
Drücken nach rechts	—	Fähre nach rechts
Betätigen des Feuerknopfes	—	Fähre bremst ab.

Auf der rechten Seite sehen Sie beim Spielen das Anzeigenfeld. In diesem Feld selbst sehen Sie die Treibstoffanzeige der Fähre und die Geschwindigkeitsanzeige. Die Geschwindigkeitsanzeige ist in drei Zonen gegliedert, in die rote, gelbe und schwarze Zone. Wenn der Pfeil auf die rote Zone zeigt, sinkt die Fähre zu schnell ab und die Landegeschwindigkeit ist zu hoch. Wenn der Pfeil auf die gelbe Zone zeigt, steigt die Fähre. Zeigt der Pfeil auf die schwarze Zone, ist die Landegeschwindigkeit richtig, und Sie werden heil landen.

Dies gilt sowohl für das Landen auf dem Planeten (Landeplätze) als auch für das Andocken im Schiff.

Außerdem sehen Sie im Anzeigefeld oben in der Mitte unter dem Wort ALPHA V einen Kreis mit einem Punkt in der Mitte. Während des Spieles ist der Punkt weiß, wenn Sie auf dem Planeten gelandet sind, braun, im Schiff schwarz.

5/5.2.2

Programmbeschreibung

0 — 5	Bildschirm vorbereiten
5 — 51	Hauptprogramm
1000 — 1200	Bild 1
2000 — 2300	Bild 2
3000 — 3300	Bild 3
4000 — 4300	Bild 4
5000 — 5200	Zeichnen des Schiffes Delta 1
6000 — 6200	Zeichnen der Anzeige
7000 — 7300	Explosionsgeräusch
7400 — 7499	Düsengeräusch
9000 — 9099	Position der Sprites festlegen
9100 — 9130	Landung auf dem Planeten
9200 — 9900	Landung im Schiff
11000 — 11054	Zeichnen des Titels und Spielen der Titelmelödie
11055 — 11900	Kurze Anleitung
12000 — 12020	Melodie
15000 — 15030	Ende des Spieles

15040 — 15900	Anzeigen der erreichten Punkte
16000 — 16100	Vorgeschichte
17000 — 17200	Abfliegen des Schiffes Delta 1
30000 — 30412	Definition der Sprites
30500 — 31000	Definition der Melodie
50000 — 50043	Daten der Sprites
50050 — 50055	Daten der Musik
60000 — 60015	Copyright

Nach Starten des Programmes werden zuerst die Sprites und die Melodie definiert. Danach wird der Titel angezeigt, und die Titelmelodie erklingt. Nach Drücken der SPACE-Taste Space können Sie wählen, ob Sie das Spiel sofort beginnen oder zuvor noch die Anleitung sehen wollen. Am Ende wird der Punktestand angezeigt.

Hier noch ein paar Tips:

— Lassen sie keine REM-Zeilen weg, denn sie werden angesprungen.

— Für andere Änderungen ist nur noch 1 KByte frei.

— Wenn Ihnen ein Bild nicht mehr gefällt, ändern Sie es einfach.

5/5.2.3

Variablenübersicht

BO	Bonus
J	Wert am Control-Port für Joystick
LG	Landegeschwindigkeit
MX	Basis-Koordinate für Sprites
MY	Basis-Koordinate für Sprites
N	Nummer des Bildes
BO	Bonus
PU(1)	Punkte
RD	Zahl der Runden

SI	Basis-Adresse SID
SL	Zahl der sicheren Landungen
V	Basis-Adresse VIC
X	X-Koordinate Sprite 0
X1	X-Koordinate Sprite 1
X2	X-Koordinate Sprite 2
Y	Y-Koordinate Sprite 0
Y1	Y-Koordinate Sprite 1
Y2	Y-Koordinate Sprite 2

Das Programmm befindet sich auf der Grundwerksdiskette

Teil 6

Weitere Programmiersprachen

6/2

PASCAL (Oxford-PASCAL)

Zu den Programmierprachen, die auf dem Commodore 64 erhältlich sind, gehört auch die Sprache PASCAL. Sie hat gerade in letzter Zeit auch im Homecomputerbereich eine weite Verbreitung gefunden. Dies ist vor allem dem Turbo-PASCAL-Compiler der Firma Borland International zu verdanken, der auch für den Anfänger leicht zu bedienen und außerdem recht preiswert ist. Turbo-PASCAL benötigt jedoch das Betriebssystem CP/M oder MS-DOS und ist daher für die meisten Commodore-64-Besitzer uninteressant. Besitzer des C 128 können diesen Compiler einsetzen.

Es gibt jedoch auch eine Reihe von PASCAL-Compilern, die ohne eine Erweiterung auf dem C 64 laufen. Einer dieser Compiler ist **Oxford-PASCAL**. Dieser Compiler hat eine beachtliche Verbreitung gefunden. Die Programmiersprache wird daher hier am Beispiel des Oxford-PASCAL vorgestellt.

Wenn Sie einen anderen PASCAL-Compiler besitzen, so können Sie die Sprache jedoch trotzdem mit diesem Kurs erlernen. Sie müssen dann jedoch die Bedienungshinweise in Ihrem Handbuch genau beachten, da hier Abweichungen auftreten können.

Außerdem verfügen die verschiedenen PASCAL-Compiler über Erweiterungen zum Standard-Wortschatz der Sprache, die von Version zu Version verschieden ist. Wo solche Abweichungen möglich sind, werden wir im Text darauf hinweisen. Hauptsächlich soll in diesem Kurs jedoch auf den Grundwortschatz von PASCAL eingegangen werden.

6/2.1

Das Sprachkonzept

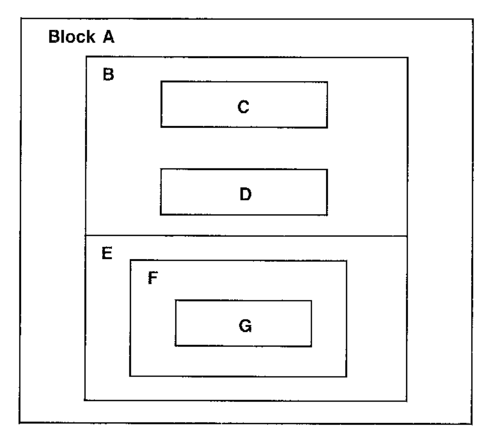
Die Sprache PASCAL wurde Anfang der siebziger Jahre von Niklaus Wirth an der Eidgenössischen Technischen Hochschule (ETH) Zürich entwickelt. Ihr Name ist im Gegensatz zu vielen anderen Namen von Programmiersprachen keine Abkürzung, sondern der Name des französischen Philosophen und Mathematikers Blaise Pascal (1623-1662).

PASCAL-Programme weisen eine Reihe von Unterschieden zu Basic-Programmen auf. Die Sprache zwingt zur **Strukturierung** des Programms. Dies bedeutet, daß Sie nicht wie in Basic einfach ein Programm eintippen können, sondern es zuerst planen müssen. So müssen alle im Programm verwendeten Variablen erst definiert werden.

Auch sind die Variablentypen wesentlich vielfältiger. Neben den Standardtypen für ganze Zahlen und Gleitkommazahlen, die Sie bereits aus Basic kennen, kann man in PASCAL eigene Variablentypen definieren oder aus bereits definierten Typen neue zusammensetzen. Auch Mengen sind in PASCAL enthalten.

Außerdem sind Abläufe wie Schleifen und Bedingungen wesentlich mächtiger als in Basic. Insbesondere ist es möglich, Programme völlig ohne den Befehl 'GOTO' zu schreiben. Das 'Springen' im Programm wird so vermieden und dadurch lesbarer und übersichtlicher.

Im Gegensatz zu Basic hat PASCAL eine **Blockstruktur**. PASCAL-Programme setzen sich im allgemeinen aus mehreren Blöcken zusammen. Ein Block kann als eigenständiges Programm aufgefaßt werden, das innerhalb eines anderen Programmes steht. Im einfachsten Fall besteht ein PASCAL-Programm also aus einem Block. Die Blockstruktur soll mit der folgenden Graphik verdeutlicht werden:



Dargestellt ist ein PASCAL-Programm, das aus den Blöcken A bis G besteht. Jeder Block löst ein Teilproblem der Gesamtaufgabe. Den Blöcken werden dabei eigene Variablen **(lokale Variablen)** zugeteilt, so daß sie andere Blöcke nicht stören können. Die gemeinsame Nutzung von Variablen ist auch möglich **(globale Variablen)**. Hierbei können Blöcke auf Variablen des Blockes, in dem sie stehen, zugreifen. Also:

Eine Variable aus Block:	! kann in folgenden Blöcken ! benutzt werden:
A	! A, B, C, D, E, F, G
B	! B, C, D
C	! C
D	! D
E	! E, F, G
F	! F, G
G	! G

Das Arbeiten mit den Blöcken werden wir im Kapitel 6/2.7 (Unterprogrammtechnik) kennenlernen.

Wie Sie sehen, bietet PASCAL eine Menge von neuen Möglichkeiten, von denen man als Basic-Programmierer nur träumen kann. Dies gilt vor allem für die Basic-Version des C 64. Allerdings erfordern PASCAL-Programme auch mehr Disziplin bei der Planung. Wir wollen uns nun den Aufbau von PASCAL-Programmen an einfachen Beispielen ansehen.

6/2.2

Einfache Beispiele

An einem einfachen Beispiel wollen wir uns den Aufbau eines PASCAL-Programms und das Arbeiten mit dem PASCAL-System ansehen. Die Systembefehle gelten dabei für Oxford-PASCAL. Sehen Sie in Ihrem Handbuch nach, falls Sie ein anderes PASCAL verwenden.

PASCAL-Programme haben im einfachsten Fall folgenden Aufbau:

Programmkopf
Konstantenvereinbarung
Variablenvereinbarung
Anweisungsteil

In unserem Beispiel wollen wir die Wurzeln der Zahlen 1 bis 5 ausgeben. Dies leistet das folgende Programm:

```
PROGRAM wurzeln;                  (* Programmkopf           *)
CONST anfang=1; ende=5;           (* Konstantenvereinbarung *)
VAR   x: INTEGER;                 (* Variablenvereinbarung  *)
      y: REAL;
BEGIN                             (* Anweisungsteil         *)
   FOR x:=anfang TO ende
    DO BEGIN
      y:=SQRT(x);
      WRITELN(x,y);
      END;
END.
```

Alle in PASCAL reservierten Worte sind im Beispiel großgeschrieben. Sie brauchen dies beim Eingeben des Programms nicht beachten.

Das Beispiel zeigt bereits eine Reihe von Eigenschaften von PASCAL. So beginnt jedes PASCAL-Programm mit dem Wort ‘PROGRAM’, gefolgt von einem Programmnamen. Dann folgt ein Semikolon. Das Semikolon hat in PASCAL etwa die gleiche Funktion wie der Doppelpunkt in Basic: es trennt Befehle. Aber in PASCAL ist die Trennung auch bei einem Übergang auf eine neue Zeile nötig, da hier, im

Gegensatz zu Basic, Befehle in der nächsten Zeile fortgesetzt werden können. Diese Eigenschaft werden Sie spätestens bei den IF-THEN-Befehlen schätzen lernen. In Basic muß die Anweisung in eine Zeile passen, in PASCAL kann sie sich dagegen über mehrere Zeilen erstrecken. Nach einem PASCAL-Wort wie 'PROGRAM', 'FOR', 'TO' oder anderen muß übrigens immer ein Leerzeichen stehen, damit PASCAL die Worte von Variablen unterscheiden kann. Variablen wie 'FORTODO' sind nämlich in PASCAL erlaubt.

Kommentare stehen zwischen '(*' und '*)' und können überall stehen, wo Leerzeichen erlaubt sind.

Es folgt die Vereinbarung von Konstanten. Sie wird durch das Wort 'CONST' eingeleitet. Dann können beliebig viele Vereinbarungen der Form '<name> = <wert>;' folgen. Enthält das Programm keine Konstanten, so kann der Teil fehlen. Namen können in PASCAL beliebig lang sein, jedoch erkennt Oxford-PASCAL nur die ersten acht Zeichen. Außerdem sind in Namen neben Buchstaben und Zahlen auch das Zeichen '_' erlaubt (Sie erreichen es durch Drücken der Commodore-Taste zusammen mit der '@'-Taste). Dieses Zeichen wird statt Leerzeichen eingesetzt, da diese in Namen nicht vorkommen dürfen.

Als nächstes werden alle im Programm verwendeten Variablen vereinbart. Dieser Teil wird durch das Wort 'VAR' eingeleitet. Dann folgt der Name einer Variablen oder eine Liste von Namen durch Kommata getrennt. Nach den Namen muß der Typ der Variable mit ':' abgetrennt angegeben werden. Zulässige Typen sind z.B. 'INTEGER' für ganze Zahlen oder 'REAL' für Fließkommazahlen. Näheres zu den Typen finden Sie in Kapitel 6/2.3.

Beispiel einer zulässigen Variablenvereinbarung:

```
VAR a, b, c, d, e: INTEGER;
    abc, dies_ist_eine_lange_variable: REAL;
    v14: INTEGER;
```

Nach den Vereinbarungen folgt schließlich der Programmteil. Es wird mit dem Wort 'BEGIN' eingeleitet und endet immer mit 'END.' Beachten Sie den Punkt hinter 'END.' Er wird oft vergessen. Das Programm ist in unserem Beispiel eine einfache FOR-Schleife, wie wir sie bereits aus Basic kennen. In PASCAL erfolgt die Zuweisung von Variablen nicht mit '=', sondern mit ':=', um die Zuweisung von der Gleichheit abzuheben. Außerdem wird die FOR-Schleife nicht mit NEXT abgeschlossen. Die FOR-Schleife sieht in PASCAL so aus:

```
FOR <variable>:= <anfangswert> TO <endwert> DO <befehl>;
```

In der Schleife darf also nur ein Befehl stehen. In PASCAL zählt hierzu aber auch eine Befehlsfolge, die in 'BEGIN' und 'END' eingeschlossen sind. Dies haben wir in unserem Beispiel ausgenutzt. In der FOR-Schleife stehen also zwei Befehle. Im

```
PROGRAM kugel;                           (* Programmkopf             *)
CONST pi=3.14159265;                     (* Konstantenvereinbarung   *)
VAR r,                                   (* Variablenvereinbarung    *)
      volumen,
      fläche: REAL;

BEGIN                                    (* PROGRAMMTEIL             *)
   WRITE ('Kugelradius:');               (* Radius einlesen          *)
   READLN (r);
   volumen:=4.0/3.0*pi*r*r*r;            (* Volumen berechnen        *)
   fläche:=4*pi*SQR(r);                  (* Fläche berechnen         *)
   WRITELN ('Volumen:',volumen:10:7);    (* und                      *)
   WRITELN ('Fläche:',flaeche:10:7);     (* Ausgeben                 *)
END. (* kugel *)
```

Beachten Sie, daß PASCAL die Potenzierung nicht kennt. Sie kann aber z.B. durch den Ausdruck 'EXP(b*LN(a))' für 'a^b' ersetzt werden.

Starten Sie nun das Programm mit 'r'. Wie Sie sehen, wird das Programm von Oxford-PASCAL erst wieder compiliert und dann gestartet. Sie werden nun vom Programm aufgefordert, einen Wert einzugeben. Für diese Eingabe ist die Zeile 'READLN (r);' zuständig, wobei 'READLN' etwa dem Basic-Befehl 'INPUT' entspricht. Geben Sie ein paar Zahlen ein und beachten Sie, wie die Ergebnisse in das Ausgabefeld geschrieben werden.

Merke: **Für die Eingabe gibt es die Befehle 'READ' und 'READLN'.**

'READ' liest Werte einer Reihe, wobei die Werte durch Leerzeichen oder Zeilenenden getrennt werden müssen.

'READLN' liest Werte, die mit <RETURN> abgeschlossen werden müssen.

Für die Ausgabe gibt es die Befehle 'WRITE' und 'WRITELN'.

'WRITE' gibt alle Werte in der Zeile aus und bleibt in der Zeile.

'WRITELN' gibt alle Werte in der Zeile aus und geht dann in die nächste Zeile.

Das Ausgabeformat kann angegeben werden und wird bei zu kleinem Feld selbsttätig erweitert.

Weitere Beispiele für Programme werden Sie bei den Befehlen kennenlernen.

ersten Befehl wird der Variable 'y' der Wert der Wurzel von x zugewiesen (y:=SQRT(x)). Die PASCAL-Funktion 'SQRT' liefert also die Wurzel einer Zahl, und entspricht somit der Basic-Funktion 'SQR'. Aber in PASCAL gibt es auch eine Funktion 'SQR', die jedoch das Quadrat der Zahl berechnet (SQR(x) = x*x engl. square). Hier ist Vorsicht geboten. Eine Übersicht über die PASCAL-Funktionen finden Sie in Kapitel 6/2.9.2.

Dann werden die Werte von x und y ausgegeben. Dies erledigt der Befehl 'WRITELN', der hier etwa dem Basic-Befehl 'print' entspricht. 'WRITELN' kann aber noch mehr, wie wir später sehen werden.

Um das Programm nun auf dem C 64 laufen zu lassen, müssen wir es zunächst eingeben. Oxford-PASCAL benutzt einen erweiterten Basic-Editor. Der Editor benötigt daher Zeilennummern, die für das PASCAL-Programm keine Bedeutung haben.

Geben Sie die erste Zeile ein:

```
10 PROGRAM wurzeln;          (* Programmkopf *)
```

Oxford-PASCAL gibt die nachfolgenden Zeilennummern nun vor:

```
20  CONST anfang=1; ende=5;          (* Konstantenvereinbarung  *)
30  VAR   x: INTEGER;                (* Variablenvereinbarung   *)
40        y: REAL;
50  BEGIN                            (* Anweisungsteil          *)
60     FOR x: anfang TO ende
70       DO BEGIN
80            y:=SQRT(x);
90            WRITELN(x,y);
100         END;
110 END.
```

Nun muß das Programm compiliert werden, d.h. es muß in Maschinensprache übersetzt werden. PASCAL ist im Gegensatz zu Basic kein Interpreter, bei dem die Zeilen während des Programmlaufs interpretiert werden müssen, sondern ein Compiler. Bei einem Compiler wird vor dem Programmlauf das Programm übersetzt. Oxford-PASCAL erzeugt jedoch wie viele PASCAL-Compiler keinen echten Maschinencode, sondern einen schnellen Zwischencode. Das Compilieren wird durch Eingabe von 'l' gestartet. Dabei wird das Programm zusammen mit Fehlermeldungen noch einmal ausgegeben. Ist alles gutgegangen, kann das Programm nun mit 'r' gestartet werden. Wir hätten allerdings auch gleich 'r' eingeben können. Oxford-PASCAL hätte den Fehler gemerkt und das Programm zuerst compiliert.

Unser Programm liefert uns folgende Ausgabe:

```
1 1.00000E+00
2 1.41421E+00
3 1.73205E+00
4 2.00000E+00
5 2.23607E+00
```

Hätten Sie dieses Ergebnis erwartet? Ein Basic-Programm hätte die Zahlen sicherlich anders ausgegeben. PASCAL verwendet immer die Exponentialschreibweise, wie sie Basic z.B. bei sehr großen oder sehr kleinen Zahlen ebenfalls verwendet. Sie können dies aber auch ändern. In PASCAL können Sie das Ausgabeformat im Gegensatz zu Basic angeben. Dies geschieht im 'WRITELN'-Befehl. Ändern Sie den Befehl wie folgt ab:

```
WRITELN(x:3,y:6:3);
```

Dabei bedeutet:

- x:3 Schreibe die Zahl x rechtsbündig in ein Feld der Länge 3.
- y:6:3 Schreibe y rechtsbündig in ein Feld der Länge 6 mit 3 Nachkommastellen.

Das Programm liefert nun folgende Ausgabe:

```
1 1.000
2 1.414
3 1.732
4 2.000
5 2.236
```

In PASCAL lassen sich mit Hilfe der Formatangabe leicht Tabellen sauber auf Bildschirm oder Drucker erstellen. Versuchen Sie zum Vergleich dies mit dem Basic des C 64 zu erreichen.

Als zweites Beispiel wollen wir uns noch die Berechnung von Volumen und Fläche einer Kugel ansehen, um uns weiter mit der Ein- und Ausgabe vertraut zu machen. Die zugehörigen Formeln lauten:

$$\text{Volumen} = \frac{4}{3} * PI * r^3$$

$$\text{Fläche} = 4 * PI * r^2 \qquad \text{C } PI=3.14159)$$

Speichern Sie das alte Programm mit 'put wurzeln' auf Diskette und löschen Sie es dann im Rechner mit 'new'. Jetzt können Sie unser neues Programm eingeben:

6/2.3

Variablen

Als nächstes wollen wir uns die Variablen, die in PASCAL möglich sind, ansehen. PASCAL bietet neben den Standardvariablen, wie Sie sie auch aus anderen Programmiersprachen (z.B. BASIC) kennen, auch die Möglichkeit, eigene Typen zu definieren. Doch zunächst zu den Standardtypen.

6/2.3.1

Standardvariablentypen

PASCAL sieht vier Standardvariablentypen vor. Die einfachste dieser Variablentypen ist der Typ **'CHAR'**. Eine Variable vom Typ 'CHAR' kann genau ein Zeichen aus dem Commodore-Zeichensatz aufnehmen. Variablen vom Typ 'CHAR' verhalten sich etwa wie Strings der Länge 1 in Basic. Neben den Zuweisungen (z.B. a:='d') gibt es vier Operationen auf dem Typ:

ORD()	ergibt die Ordnungszahl eines Zeichens. Diese Funktion entspricht in Basic 'ASC()'. Bsp.: ORD ('0')=48.
CHR()	ist die Umkehrfunktion zu ORD(). In Basic: 'CHR$()'. Bsp.: CHR(48)='0'.
SUCC()	gibt den Nachfolger eines Zeichens an. Bsp.: SUCC('A')='B'.
PRED()	gibt den Vorgänger an. Bsp.: PRED('B')='A'.

Ein weiterer Datentyp ist der Typ **'BOOLEAN'**. Hierzu gibt es in Basic keinen entsprechenden Typ. Variablen vom Typ 'BOOLEAN' können nur die in PASCAL definierten Werte **'TRUE'**, oder **'FALSE'** annehmen. Werte von diesem Typ ergeben sich z.B. bei Vergleichen:

```
'a' = 'b'   ergibt false
'a'<'b'     ergibt true
5>10        ergibt false
```

Auf dem Typ sind neben der Zuweisung (z.B. 'ende:=false') drei logische Operationen erlaubt: **'AND'**, **'OR'** und **'NOT'**, die auch aus Basic bekannt sind.

X	Y	X AND Y	X OR Y	NOT X
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Hier sei allerdings auf die etwas ungünstige Prioritätenverteilung hingewiesen. In PASCAL gelten folgende Prioritäten:

```
   höchste Priorität: NOT
               dann: *,/,DIV,MOD,AND
               dann: +,—,OR
niedrigste Priorität: =,<,>,<=,>=,<>
```

Die Basic-Anwendung 'IF a<12 AND b>54 THEN . . . ' muß in PASCAL daher als 'IF (a<12) AND (b>54) THEN . . . ' geschrieben werden.

Das folgende kleine Beispiel soll den Umgang mit den Typen 'CHAR' und 'BOOLEAN' zeigen. Es gibt die Zeichen 'a' bis 'z' auf dem Bildschirm in Zeilen zu je fünf Zeichen aus:

```
PROGRAM  a__bis__z;
CONST    anfang  ='a';           (* von Buchstabe a           *)
         ende    ='z';           (* bis Buchstabe z           *)

VAR      buchstabe: CHAR;        (* aktueller Buchstabe       *)
         neue zeile: BOOLEAN;    (* Entscheidung für neue Zeile *)
         i:         CHAR'        (* Zählt die Buchstaben      *)
```

```
BEGIN                          (* Programmstart                    *)
  i:='0';                      (* Zähler auf '0'                   *)
  FOR buchstabe := anfang TO ende DO
                               (* FOR Schleife auf Typ CHAR von
                               'a' bis 'z'                         *)
    BEGIN
      WRITE (buchstabe:1);     (* buchst. in Feld der Länge 1      *)
      i:=SUCC(i);              (* Zähler erhöhen                   *)
      neue_zeile:=i='6';       (* neue Zeile bei i='9'             *)
      IF neue_zeile TEN
        BEGIN                  (* neue Zeile                       *)
        WRITELN;               (* und Zähler rücksetzen            *)
        i:='0';
      END;                     (* der IF-Bedienung                 *)
    END;                       (* der FOR-Schleife)                *)
END.                           (* Programmende                     *)
```

Wie Sie sehen, kann man FOR-Schleifen mit Variablen vom Typ CHAR bilden und so sehr einfach die Buchstaben durchlaufen. In Basic müßten Sie eine spezielle Laufvariable vorsehen und eine Typumwandlung vornehmen, um das gleiche zu erreichen. Beachten Sie bitte auch, daß hier SUCC auf eine Variable von Typ INTEGER verwendet wird. SUCC hat dann die gleiche Funktion wie die Addition von eins. Allgemein gilt: SUCC, PRED und ORD arbeiten auf allen aufzählbaren Datentypen (also nicht auf REAL!).

Die beiden anderen Datentypen **'INTEGER'** und **'REAL'** haben wir bereits kennengelernt. Der Datentyp 'INTEGER' umfaßt ganze Zahlen im Bereich von —32768 bis +32767. Er entspricht den Basic-Variablen, die mit '%' gekennzeichnet sind (z.B.: ab%). Auf ihm sind die Operationen '+', '-', und '*' wie gewohnt erlaubt. Etwas anders ist es bei der Division. Da eine Division i.a. einen Quotienten und einen Rest ergibt, gibt es die zwei Operationen **'DIV'** und **'MOD'**, wobei 'DIV' den Quotienten und 'MOD' den Rest liefert.

Bsp.: 1234/56 ergibt 22 Rest 2. Folglich liefern die Pascal Ausdrücke
1234 DIV 56 als Ergebnis 22
1234 MOD 56 als Ergebnis 2.

Die Grenze des Bereichs ist in PASCAL als Konstante **'maxint'** (=32767) vordefiniert.

Variablen vom Typ **'REAL'** kennen Sie bereits aus Basic. Auf ihnen sind die Operationen '+', '-', '*' und '/' wie gewohnt definiert und liefern die gleichen Ergebnisse wie in Basic auch.

6/2.3.2

Definition eigener Typen

Interessant an PASCAL ist, daß die Sprache — im Gegensatz zu Basic — die Definition eigener Datentypen erlaubt. So ist es zum Beispiel möglich, einen Typ zu definieren, der die Wochentage enthält. Dies wird durch Aufzählung erreicht:

```
TYPE tage = (mo,di,mi,do,fr,sa,so)
```

Die Definition wird mit dem Wort 'TYPE' eingeleitet. Dann folgt der Name des Typs. In unserem Beispiel ist es 'tage'. Nun müssen die Elemente des Typs nur noch aufgezählt werden. Durch die Aufzählung definieren Sie gleichzeitig eine Ordnung auf den Typ, d.h. mo<di<mi<do<fr<sa<so. Daher sind die Operationen 'ORD', 'SUCC' und 'PRED' sowie die Verwendung in FOR-Schleifen mit selbstdefinierten Typen erlaubt.

Eine besondere Form der Aufzählung von Typen ist der Teilbereich. Angenommen, Sie wollen einen Typ definieren, der ein Unterbereich eines bereits bekannten Typs ist, z.B. die Zahlen 125 bis 1024. Dann müßten Sie diese aufzählen:

```
TYPE teil = (125,126,127,128,129,130,131,132 . . .)
```

Da dies kaum möglich ist, gibt es eine Abkürzung:

```
TYPE teil = 125..1024;
```

Zulässige Typendefinitionen sind also z.B.:

```
TYPE  früchte = (apfel, birne, banane);
      byte = 0..255;
      tage = (mon,die,mit,don,fre,sam,son);
          arbeitstage = mon..fre;
          wochenende = sam..son;
```

Beachten Sie, daß die Typen arbeitstage und wochenende erst nach dem Typ tage definiert werden können, da sie Unterbereichstypen von tage sind.

Unterbereichstypen können bei der Variablendefinition auch direkt angegeben werden.

Das folgende Beispiel zeigt den Einsatz von Unterbereichstypen. Weitere Anwendungen werden Ihnen in Kapitel 6/2.8 Strukturierte Datentypen begegnen, da wir dann erst PASCAL richtig einsetzen können.

```
PROGRAM datum;
(* Berechnet das morgige Datum im Zeitraum von 1.1.1901 bis
   30.12.1999                                                    *)
TYPE      tage     = 1..31;    (* Typendefinition des Datums     *)
          monate   = 1..12;
          jahre    = 1..99;
VAR       tag:     tage;       (* Variablendefinition            *)
          monat:   monate;
          jahr:    jahre;
          laenge:  28..31;     (* Direkte Angabe eines Types     *)
```

```
BEGIN
  WRITELN ('Datum eingeben:');            (* Datum einlesen       *)
  WRITE   ('Tag:      '); READLN (tag);
  WRITE   ('Monat;  '); READLN (monat);
  WRITE   ('Jahr:    '); READLN (jahr);
  laenge:=31;                              (* Monatslänge berechnen *)
  IF (monat=4) OR (monat=6) OR (monat=9) OR (monat=11)
  THEN laenge:=30                          (* Monate mit 30 Tagen  *)
  IF monat=2 THEN                          (* Februar gesondert    *)
    IF (jahr MOD 4=0) THEN laenge:=29      (* Schaltjahr           *)
                      ELSE laenge:=28;
  IF tag<laenge
     tag:=tag+1                            (* nur Tag ändern       *)
  THEN ELSE BEGIN
              tag:=1;                      (* neuer Monat          *)
              IF monat=12
                THEN BEGIN
                  monat:=1;                (* neues Jahr           *)
                  jahr:=jahr+1
                END
               ELSE monat:=monat+1;        (* nur neuer Monat      *)
     END;
  WRITELN ('Morgen ist der',tag:1,'.',monat:1,'.',
           jahr+1900:1);                   (* morgiges Datum
                                              ausgeben             *)
END.
```

6/2.4

Strukturbefehle

6/2.4.1

Die GOTO-Anweisung

PASCAL kennt genau wie Basic die GOTO-Anweisung. Gesprungen wird jedoch nicht zu einer Zeilennummer, sondern zu einem Label (Sprungmarke), das aus einer 8stelligen positiven ganzen Zahl besteht (in Standard-PASCAL 4 Stellen). Ein Label kann vor jeder PASCAL-Anweisung stehen und wird durch einen Doppelpunkt von der Anweisung abgetrennt:

Beispiel:
```
12: x:=i*56.7;
IF x <> 256 THEN GOTO 12;
```

Die in einem Programm verwendeten Label müssen genau wie die Variablen erst vereinbart werden. Dies geschieht durch die Anweisung **'LABEL'**, gefolgt von einer Liste von Labeln, die durch ein Komma getrennt sind. Die Labeldefinition folgt direkt dem Programmkopf.

Die Anweisung GOTO sollte möglichst vermieden werden, da sie ein Programm unleserlich macht. Durch die Blockstruktur von PASCAL ist 'GOTO' auch unnötig. Sie hat jedoch durchaus einen Sinn. So kann z.B. in einem Fehlerfall mit Hilfe von GOTO einfach aus einer Struktur ausgebrochen werden. Ein Beispiel für den Gebrauch von GOTO soll hier gezeigt werden. Es gibt die Zahlen 1 bis 5 sowie deren Quadrate aus.

```
PROGRAM goto_demo;
LABEL 1;                          (* Label vereinbaren        *)
VAR i: INTEGER;
BEGIN
  i:=0;
  1: i:=i+1;                      (* Hier ist das Label gesetzt *)
  WRITELN (i, i*i);
  IF i<5 THEN GOTO 1;             (* Sprung zum Label         *)
END.
```

6/2.4.2

IF . . . THEN . . . ELSE

Die 'IF . . . THEN'-Anweisung kennen Sie bereits aus Basic und unseren Beispielen. Sie ist in PASCAL um die Anweisung 'ELSE' erweitert, die allerdings auch fehlen kann. Zulässige Befehlsformen sind also:

```
IF <Bool'scher Ausdruck> THEN <Befehl 1> ELSE <Befehl 2>
IF <Bool'scher Ausdruck> THEN <Befehl>
```

Die Befehle können dabei wieder Strukturen, in 'BEGIN' und 'END' eingeklammert sein. Ist der Bool'sche Ausdruck wahr ('true'), so wird die Befehlsfolge hinter 'THEN' ausgeführt, anderenfalls die Befehlsfolge hinter 'ELSE', falls 'ELSE' angegeben ist. Sonst wird mit den nachfolgenden Befehlen fortgefahren.

Beispiel: Es sollen zwei Zahlen a und b eingelesen und ausgegeben werden, ob a gleich b, oder kleiner, oder größer ist.

```
PROGRAM vergleich;
VAR a,b: REAL;

BEGIN                                          (* Zahlen a und b einlesen *)
  WRITE ('Geben Sie die erste Zahl ein: ');
  READLN (a);
  WRITE ('Geben Sie die zweite Zahl ein: ');
  READLN (b);

  IF a=b
    THEN WRITELN ('a ist gleich b')
    ELSE BEGIN
           IF a < b THEN WRITELN ('a ist kleiner b')
                    ELSE WRITELN ('a ist großer b')
           END;
END.
```

Beachten Sie, daß vor 'ELSE' kein Semikolon stehen darf!

6/2.4.3

Die CASE-Anweisung

Mit Hilfe der IF ... THEN ... ELSE-Konstruktion lassen sich auch lange Abfragen z.B. für Menüs nach folgendem Schema konstruieren:

```
     IF <bedingung 1> THEN <befehl 1>
ELSE IF <bedingung 2> THEN <befehl 2>
ELSE IF <bedingung 3> THEN <befehl 3>
ELSE IF <bedingung 4> THEN <befehl 4>
etc.
```

Da dies aber sehr umständlich und unleserlich ist, gibt es in PASCAL eine Anweisung, die solche Strukturen vereinfacht: die CASE-Anweisung. Sie hat folgende Form:

```
CASE <ausdruck> OF
        <Konstantenliste 1> : <befehl 1>;
        <Konstantenliste 2> : <befehl 2>;
        <Konstantenliste 3> : <befehl 3>;
                          ''
                          ''
        <Konstantenliste n> : <befehl n>;
END;
```

Dabei ist der <ausdruck> ein Ausdruck von Typ INTEGER, CHAR oder BOOLEAN. Die Konstanten müssen vom gleichen Typ wie der Ausdruck sein. Als Konstantenliste ist eine solche Konstante oder mehrere solcher Konstanten durch Komma getrennt zulässig.

Die CASE-Anweisung arbeitet dann wie folgt: Der Ausdruck <ausdruck> wird berechnet. Das Ergebnis wird mit den Konstantenlisten verglichen. Wird das Ergebnis in einer Liste gefunden, so wird der Befehl hinter der Liste ausgeführt. Steht das Ergebnis in keiner Liste, wird (in Oxford-PASCAL) ein Fehler gemeldet.

Als Beispiel wollen wir ein kleines Taschenrechnerprogramm erstellen, das die Befehle '+', '-', '*', und '/' versteht.

```
PROGRAM rechner;
VAR operation: CHAR;
    arg1,arg2,
    ergebnis : REAL;
BEGIN
  WRITELN ('Taschenrechner');
  WRITE ('arg1:'); READLN (arg1);
  WRITE ('op: '); READLN (operation);
  WRITE ('arg2: '); READLN (arg2);
  CASE operation OF
  '+' :  ergebnis:=arg1+arg2;
  '-' :  ergebnis:=arg1—arg2;
  '*' :  ergebnis:=arg1*arg2;
  '/' :  BEGIN
           IF arg2<>0 THEN ergebnis:=arg1/arg2
                      ELSE WRITELN ('Division durch Null')
         END;
  END; (* of Case *)
  WRITELN (arg1:9:2,op:1,arg2:9:2,'=',ergebnis:9:2);
END.
```

6/2.5

Schleifen

Neben den Strukturbefehlen gehört die Schleifenbildung zu den häufigsten Strukturen in Programmen. Eine dieser Schleifen, die FOR-Schleife, kennen Sie bereits von Basic-Programmen. PASCAL kennt neben dieser Schleife aber noch zwei weitere Schleifen, welche die Möglichkeiten wesentlich erweitern.

6/2.5.1

FOR . . TO . . DO

Die FOR-Schleife haben wir bereits im Kapitel 6/2.2 'Einfache Beispiele' kennengelernt. Sie hat in Pascal die folgende Form:

```
FOR<Variable>:= <Anfangswert> TO <Endwert> DO <Befehl>;
```

Der Variablen wird der Anfangswert zugewiesen. Ist die Variable kleiner oder gleich dem Endwert, wird der Befehl ausgeführt. Dann wird die Variable um eine Einheit erhöht, und die Schleife von neuem durchlaufen. Nach dem Verlassen der Schleife hat die Variable den Wert <endwert> +1. Eine weitere Möglichkeit besteht in der folgenden Form:

```
FOR <Variable>:= <Anfangswert> DOWNTO <Endwert> DO <Befehl>;
```

Der Variablen wird wieder der Anfangswert zugewiesen. Hier muß die Variable jedoch größer oder gleich dem Endwert sein, damit der Befehl ausgeführt wird. Dann wird die Variable um eine Einheit erniedrigt, und die Schleife von neuem durchlaufen. Nach dem Verlassen der Schleife hat die Variable den Wert <endwert>-1. Eine Schrittangabe, wie sie der STEP-Befehl in Basic ermöglicht, ist in PASCAL nicht vorgesehen. Es gibt also nur die Schrittweiten 1 und -1.

In dem folgenden Beispiel wollen wir die Fakultät einer Zahl berechnen. Die Fakultät einer Zahl ist das Produkt der Zahlen von 1 bis zu der Zahl selbst.

Beispiel: Berechnung von 10! (lies: zehn Fakultät)
10! = 10*9*8*7*6*5*4*3*2*1 = 3628800

```
PROGRAM fakultät_1; (* Berechnet die Fakultät einer Zahl *)

VAR i, k:INTEGER; (* Eingabezahl und Laufindex *)
    j:REAL;       (* Ergebnis                  *)

BEGIN
  WRITE ('Zahl eingeben:');            (* Zahl einlesen       *)
  READLN(i);
  j:=1;                                (* Ergebnis = 1        *)
  FOR k:=i DOWNTO 1 DO j:=j*k;         (* Fakultät berechnen  *)
  WRITELN (i:1,'!=',j:7:1);            (* Ausgabe             *)
END
```

6/2.5.2

REPEAT . . UNTIL

Eine weitere Schleife ist die REPEAT-Schleife. Sie wiederholt eine Folge von Befehlen, bis eine bestimmte Bedingung erfüllt ist. Beachten Sie, daß diese Struktur grundsätzlich verschieden von der FOR . . TO . . DO-Schleife gestaltet ist.

Die REPEAT-UNTIL-Schleife hat folgende Form:

```
REPEAT
    <Befehl 1>
    <Befehl 2>
        ''
UNTIL <Bedingung>
```

Die Befehle zwischen REPEAT und UNTIL brauchen also nicht durch BEGIN und END zusammengefaßt werden, wie das bei den anderen Schleifentypen erforderlich ist. Beachten Sie auch, daß die Befehle in der Schleife immer mindestens einmal durchlaufen werden. Auch muß der Programmierer dafür sorgen, daß die Bedingung irgendwann einmal erfüllt ist, denn sonst erhält man eine Endlosschleife. Dieser Effekt kann allerdings auch einmal erwünscht sein. So läuft z.B. eine Heizungsregelung immer in einer Endlosschleife. Diese kann dann einfach programmiert werden:

```
REPEAT
    <Befehl 1>
    <Befehl 2>
        ''
UNTIL FALSE;
```

Als kleines Beispiel wollen wir die Fakultät diesmal mit der Repeat-Schleife berechnen:

```
PROGRAM fakultät_2;
VAR i,k:INTEGER; (* ZAHL UND SCHLEIFENINDEX *)
    j:REAL;       (* ERGEBNIS                *)
BEGIN
  WRITE ('Zahl eingeben:'); (* ZAHL EINLESEN  *)
  READLN(i);
  j:=1; k:=i; (* INDEX UND ERGEBNIS AUF ANFANGSWERT *)
  REPEAT      (* FAKULTÄT BERECHNEN                 *)
      j: =j*k;
      k:=k-1
  UNTIL k<=1;
  WRITELN (i:1, '!= ',j:7:)1;    (* ERGEBNIS AUSGEBEN    *)
END.
```

Die nun folgende Schleife besitzt von der Struktur her eine gewisse Ähnlichkeit mit der REPEAT-UNTIL-Schleife. Es ist die Schleife:

6/2.5.3

WHILE . . DO

Sie ist der letzte Schleifentyp den PASCAL kennt. Die WHILE . . DO-Schleife führt einen Befehl, oder mehrere, durch BEGIN und END geklammerte Befehle, solange aus, wie eine bestimmte Bedingung erfüllt ist. Sie hat die folgende Form:

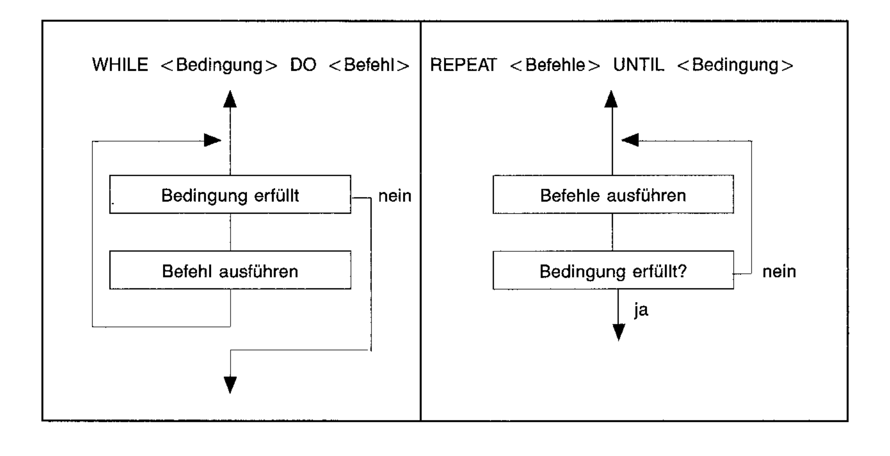
```
WHILE <Bedingung> DO <Befehl>;
```

Im Gegensatz zur REPEAT-Schleife wird der Befehl der WHILE-Schleife bei falscher Bedingung nicht ausgeführt. Aber auch hier muß der Programmierer für den Abbruch der Schleife sorgen, d.h. die Bedingung muß irgendwann einmal 'FALSE' werden. Außerdem wird hier die Bedingung nicht geprüft, nachdem die Befehle abgearbeitet wurden, sondern bereits bevor die Befehle ausgeführt werden. Dies ist ein wesentlicher Unterschied. Zwischen den Schleifen REPEAT und WHILE entscheidet man sich also, indem man bestimmt, ob die Befehle *vor* dem Prüfen der Bedingung bereits ausgeführt werden sollen.

Als Beispiel wollen wir diesmal die Quersumme einer Zahl berechnen. Die Quersumme einer Zahl ist die Summe Ihrer Ziffern. Beispiel: Die Quersumme von 83745 ist 8+3+7+4+5=27.

```
PROGRAM quersumme;
VAR n,k,s: INTEGER;    (* Zahl, Hilfsvar. und Quersumme *)
BEGIN
  WRITE ('geben Sie eine Zahl ein');
  READLN(n);           (* Zahl einlesen          *)
  s:=0;                (* Summe = 0              *)
  k:=n;                (* K = Zahl               *)
    WHILE (k<>0 DO     (* WHILE-Schleife         *)
    BEGIN
      s:=s+k MOD 10;   (* Zahl zerlegen          *)
      k:=k DIV 10;
    END;
  WRITELN ('Die Quersumme von ',n:1,' ist ',s:1);
END.
```

Die folgenden zwei Flußdiagramme machen den Unterschied zwischen der WHILE . . DO- und der REPEAT . . UNTIL-Schleife noch einmal graphisch deutlich:


WHILE <Bedingung> DO <Befehl>
Bedingung erfüllt
nein
Befehl ausführen
REPEAT <Befehle> UNTIL <Bedingung>
Befehle ausführen
Bedingung erfüllt?
nein
ja

6/2.6

Felder

In PASCAL ist natürlich auch das Arbeiten mit Feldern erlaubt. Im Gegensatz zu der Basic-Anweisung DIM, kann man in PASCAL aber nicht nur die obere Feldgrenze, sondern auch die untere angeben. In Basic ist diese bekanntlich immer Null. Außerdem kann als Feldindex jeder aufzählbare Typ verwendet werden. Ein Feld wird in PASCAL folgendermaßen vereinbart:

```
VAR <Variablenname>: ARRAY [<Bereich>] of <Typ>;
```

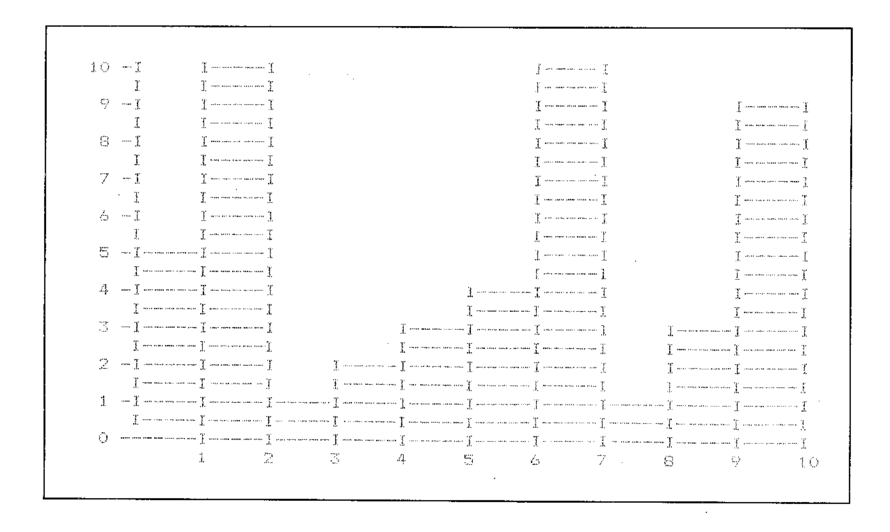
Diese Anweisung definiert ein Feld mit dem Namen <Variablenname> der Größe <Bereich>, wobei die Feldelemente von Typ <Typ> sind. Um ein Feld des Namens 'zahlen' reeller Zahlen, die mit einem Index von 18 bis 45 versehen sind, zu definieren, geben wir also folgendes an:

```
       VAR zahlen: ARRAY [18..45] of REAL;
oder:  TYP index_bereich = 18..45;
       VAR zahlen: ARRAY [index-bereich] of real;
oder:  TYP zahlenfeld = ARRAY [18..45] of real;
       VAR zahlen: zahlenfeld;
```

Um Feldelemenete im Programm ansprechen zu können, werden sie wie normale Variablen behandelt. Nur müssen Sie, wie Sie es auch von Basic gewohnt sind, den Feldindex angeben. Im Gegensatz zu Basic verwendet PASCAL jedoch eckige Klammern für den Feldindex.

Als Beispiel wollen wir ein Programm betrachten, welches die folgende Aufgabe lösen kann: Es soll zehn Zahlen zwischen Null und Zehn einlesen und sie als

Säulengraphic (also in Form eines Histogramms) ausgeben. Die Höhe der einzelnen Säulen entspricht dann den eingegebenen Werten (variiert also von 0 bis 10), während die Breite der einzelnen Säulen konstant bleibt (pro Säule sind es hier 7 Zeichen). Für die Eingabe der Werte 5, 10, 1, 2, 3, 4, 10, 1, 3, 9 würde dann die Ausgabe folgendermaßen aussehen:



Zunächst entwerfen wir den Programmkopf: Als erste Konstante wird die Anzahl der Eingabewerte plus 1 benötigt, sie soll 'intanz' heißen. Die zweite Konstante soll die Anzahl der einzugebenden Meßwerte begrenzen, wir nennen sie 'maxanz'. Um die Säulen bei der Ausgabe entstehen zu lassen (oder entsprechend Leerraum zu haben), werden als letztes noch Zeichenketten benötigt: 'full' als ausgefülltes Ausgabefeld (-----I) und 'empty' als leeres Ausgabefeld (   ).

Die restlichen benötigten Parameter sind Variablen: Wir nehmen für die jeweils aktuelle Zeile die Variable *Zeile*, für die jeweils aktuelle Spalte die Variable *Spalte*. Als Laufvariablen der FOR-Schleifen für den Programmlauf bzw. der Ausgabe nehmen wir i und k. Um die Eingabe darauf untersuchen zu können, ob die Werte im zulässigen Bereich liegen, wird noch eine boolesche Variable gültig eingeführt. Als letztes müssen die eingelesenen Zahlen noch in einem Feld abgelegt werden, das hier Anzahl heißen soll.

```
PROGRAM histogramm;
CONST intanz   =11; (* Intervallanz.+1 *)
      maxanz=10; (* max. Meßwertanzahl *)
      full     ='-----I'; (* Bildaufbau *)
      empty  ='     '; (* 5 mal Space *)
VAR   zeile:  INTEGER;  (* Zeilenindex *)
      spalte: INTEGER;  (* Spaltenindex *)
      k, i:   INTEGER;  (* Index *)
      anzahl:ARRAY [1 . . intanz] of INTEGER; (* Meßwertanzahl *)
      gültig: BOOLEAN; (* Flag für Eingabe gültig *)
```

Im Hauptprogramm wird zunächst die Variable anzahl [intanz] auf Null gesetzt. Dies brauchen wir als Hilfswert, wenn wir bei der Ausgabe derGraphik immer zwei benachbarte Werte betrachten. Dann werden die Werte in einer FOR-Schleife eingelesen. In dieser FOR-Schleife erfolgt die Eingabe der einzelnen Werte, bis die boolesche Variable 'gültig' den Wert TRUE enthält. Dies ist genau dann der Fall, wenn die Eingabe im erlaubten Bereich liegt.

```
BEGIN
  anzahl[intanz]:=0;
  FOR i:=1 TO intanz-1 DO
    REPEAT
      WRITE ('Intervall (',i-1:1,',',i:1,')');
      READLN (anzahl[i]);
      gueltig:=(anzahl[i]>=0)AND(anzahl) [i]<=maxanz);
      IF NOT gueltig THEN WRITELN ('-- Falsche Einfabe');
    UNTIL gueltig;
```

Als nächstes erfolgt die Ausgabe der Graphik. Dazu wird in einer Hauptschleife die Variable Zeile von zehn bis eins heruntergezählt. Da jeder Zahlenwert durch zwei Zeilen repräsentiert wird, wird die Variable k in einer weiteren Schleife eins bis zwei gezählt. Ist k=1 wird der Zeile der Wert vorangestellt. Dies liefert der IF . . THEN . .ELSE Ausdruck.

Für jede Zeile wird nun die Spalte durchlaufen. Ist der eingelesene Wert größer oder gleich der Zeile, muß 'I-----I' ausgegeben werden, sonst ein leeres Feld. Dabei muß am rechten Rand entschieden werden, ob das Zeichen 'I' ausgegeben werden muß, falls das Ausgabefeld leer war. Dies ist für den Übergang von leeren auf volle Felder notwendig.

```
FOR zeile:=maxanz DOWNTO 1 DO
BEGIN
  FOR k:=1 TO 2 DO
  BEGIN
    WRITELN;
    IF k=2 THEN WRITE ('    I')                    (* 4 mal Space *)
           ELSE WRITE (zeile:2,' -I');
    FOR spalte:=1 TO intanz-1 DO
    BEGIN
      IF anzahl [spalte]>=zeile
      THEN WRITE (full)
      ELSE
        BEGIN
          WRITE (empty)
          i:=spalte;
          IF (anzahl[i+1]>anzahl[i]) AND (anzahl[i+1]>=zeile)
          THEN WRITE ('I')
          ELSE WRITE (' ');                        (* 1 mal Space *)
        END
    END          (* spalten *)
  END            (* 2mal    *)
END;             (* zeilen  *)
```

Zum Schluß muß nur noch die Skala unter dem Bild gedruckt werden. Dies kann mit einfachen FOR-Schleifen erfolgen:

```
  WRITELN;
  WRITE ('     0 -I');                             (* 5 mal Space *)
  FOR i:=1 TO intanz-1 DO WRITE (full);            (* 5 mal Space *)
  WRITELN;
  WRITE ('     ');
  FOR i:=1 TO intanz-1 DO WRITE ('     ', i:1);
  WRITELN;
END.
```

Natürlich sind in PASCAL auch mehrdimensionale Felder erlaubt. Eine einfache Möglichkeit, dies zu verwirklichen, geht direkt aus der Art der Definition von Feldern hervor. Wenn man als Typangabe wieder ein Feld angibt, erhält man ein mehrdimensionales Feld. Eine entsprechende Definition sähe dann zum Beispiel so aus:

```
VAR feld : ARRAY [1 . . 10] OF
             ARRAY [20 . . 23] OF
               ARRAY [1 . . 2] OF INTEGER;
```

Um die zugehörige Variable ansprechen zu können, müssen dann natürlich auch drei Indexwerte angegeben werden. Also z.B. 'a:=feld[5] [21] [1]'. Da diese Schreibweise recht umständlich ist, kann man die drei Komponenten in der Definition und bei Arbeiten mit Feldvariablen durch ein Komma trennen. Die oben gemachten Angaben stimmen daher mit den folgenden überein:

```
      VAR feld: ARRAY [1 . . 10,20 . . 23,1 . . 2] OF INTEGER
bzw.  a:=feld [5,21,1];
```

Das folgende Programm arbeitet mit derartigen Feldern. Es betrachtet zwei einzulesende Felder a und b als Matrizen, und multipliziert diese zwei Felder zu einer Ergebnismatrix c.

Eine Matrix ist dabei eine Anordnung reeller Zahlen in einem zweidimensionalen Schema. Die folgende Matrix a heißt 3x4 Matrix (lies: drei-kreuz-vier Matrix), da sie aus drei Zeilen zu vier Spalten besteht.

$$a = \begin{pmatrix} 1 & 1 & 4 & 5 \\ 2 & 6 & 9 & 3 \\ 4 & 5 & 6 & 7 \end{pmatrix}$$

Allgemein hat a als mxn Matrix die folgende Form:

$$a = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

Man kann nun eine p*q Matrix mit einer p*r Matrix multiplizieren, und erhält nach der Formel

$$c_{ij} = a_{i1}*b_{1j}+a_{i2}*b_{2j}+a_{i3}*b_{3j}+\ldots+a_{iq}*b_{qj}$$

eine p*r Matrix.

Beispiel:

$$\begin{pmatrix} 5 & 3 & 8 & 4 \\ 2 & 1 & 0 & 4 \\ 6 & 2 & 1 & 6 \end{pmatrix} * \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 1 & 2 \end{pmatrix} = \begin{pmatrix} 58 & 78 \\ 9 & 16 \\ 23 & 38 \end{pmatrix}$$

(denn: $58=5*1+3*3+8*5+4*1$, $78=5*2+3*4+8*6+4*2$
$9=2*1+1*3+0*5+4*1$, $16=2*2+1*4+0*6+4*2$
$23=6*1+2*3+1*5+6*1$, $38=6*2+2*4+1*6+6*2$)

```
PROGRAM matrixmult;

(* multipliziert eine p*q mit einer q*r Matrix reeller Zahlen *)


CONST p=3; (* Definition der Groessen der Matrizen *)
      q=4;
      r=2;

VAR   i,j,k: INTEGER;                        (* Hilfsvariablen *)
      a:     ARRAY[1..p,1..q] OF REAL;     (* Matrix a *)
      b:     ARRAY[1..q,1..r] OF REAL;     (* Matrix b *)
      c:     ARRAY[1..p,1..r] OF REAL;     (* Matrix c *)

BEGIN                          (* Matrix a einlesen *)
  FOR i:=1 TO p DO
    FOR j:=1 TO q DO
    BEGIN
      WRITE  ('a(',i:1,',',j:1,')= ');
      READLN (a[i,j]);
    END;
  WRITELN;

                              (* Matrix b einlesen *)
```

```
FOR i:=1 TO q DO
  FOR j:=1 TO r DO
  BEGIN
    WRITE ('b(',i:1,',',j:1,')= ');
    READLN (b[i,j]);
  END;
WRITELN;

                    (* multiplizieren *)

FOR i:=1 TO p DO
  FOR j:=1 TO r DO
  BEGIN
    c[i,j]:=0;
    FOR k:=1 TO q DO c[i,j]:=c[i,j]+a[i,k]*b[k,j];
  END;

                    (* Matrix c ausgeben *)

FOR i:=1 TO p DO
  FOR j:=1 TO r DO
    WRITELN('c(',i:1,',',j:1,')= ',c[i,j]);
  END.
```

6/2.7

Unterprogrammtechnik

Unterprogramme kennen Sie bereits aus Basic: Ein Programmabschnitt kann mit GOSUB von jeder beliebigen Stelle aus aufgerufen werden, und kehrt nach RETURN an diese Stelle zurück. Auch in PASCAL ist die Unterprogrammtechnik enthalten. Sie ist jedoch wesentlich weiter entwickelt, und bietet daher mehr Möglichkeiten, als die Technik des C 64 Basic. Die von PASCAL angebotenen Möglichkeiten erlauben das Erstellen von Prozeduren (entspricht etwa dem GOSUB . . RETURN-Mechanismus des Basic) und von Funktionen (dies ist in einer sehr eingeschränkten Form mit DEF FN ebenfalls in Basic möglich).

6/2.7.1

Prozeduren

Prozeduren sind die Unterprogramme in PASCAL-Programmen. Sie werden durch das reservierte Wort 'PROCEDURE' eingeleitet. Anschließend muß der Prozedur ein Name zugewiesen werden. Nun kann ein vollständiger Programmblock folgen, der eigene Variablen, eigene Typvereinbarungen und sogar wieder eigene Prozeduren enthalten kann.

Die einfachste Form enthält also lediglich eine Folge von Befehlen, die zwischen BEGIN und END eingeschlossen sind. Die folgende Prozedur dient dem Löschen des Bildschirms:

```
PROCEDURE clrscr;
BEGIN
  WRITE (CHR(146))
END;
```

Um die Prozedur aufzurufen, wird lediglich der Prozedurname als Befehl angegeben. Dabei ist zu beachten, daß die Prozedur vor ihrem ersten Aufruf definiert werden muß. Ein vollständiges Programm, welches die Prozedur CLRSCR benutzt, sähe also etwa so aus:

```
PROGRAM beispiel;

LABEL .....
CONST .....
VAR   .....

PROCEDURE clrscr; (* Definition der Prozedur *)
BEGIN
  WRITE (CHR(146))
END;

BEGIN      (* Hier beginnt das Hauptprogramm *)
   :
   clrscr;  (* Aufruf der Prozedur *)
   :
END.
```

Den Prozeduren können auch Parameter übergeben werden. PASCAL läßt dabei zwei Arten von Parametern zu: Wertparameter und Variablenparameter. Bei Wertparametern wird einer nur in der Prozedur gültigen Variablen (einer lokalen Variablen) ein Wert übergeben. Dies kann ein beliebiger in PASCAL gültiger Ausdruck sein. Bei Variablenparametern wird der Prozedur der Name einer Variablen übergeben. Die Prozedur benutzt dann die übergebene Variable.

Die Liste der Parameter folgt dem Prozedurnamen, und wird in Klammern eingeschlossen. Jedem Parameter muß der Typ durch einen Doppelpunkt getrennt folgen. Dabei können mehrere Parameter gleichen Typs wieder durch Kommata getrennt werden.

Variablenparametern wird das reservierte Wort 'VAR' vorangestellt. Die einzelnen Parametervereinbarungen werden, wie wir von der Variablenvereinbarung gewohnt sind, durch ein Semikolon getrennt.

Versuchen Sie herauszufinden, welche Ausgabe das folgende Programm liefert:

```
PROGRAM demo;
VAR a,b,c: INTEGER;
PROCEDURE test (a:INTEGER; VAR b:INTEGER);
VAR c:INTEGER;
BEGIN
  c:=3;
  WRITELN (a,b,c);
  a:=5; b:=7;
  WRITELN (a,b,c)
END;
BEGIN
  a:=19; b:=8; c:=1
  WRITELN (a,b,c);
  test (b,a);
  WRITELN (a,b,c)
END.
```

Das Programm weist den Variable a, b und c zunächst die Werte 19, 8 und 1 zu, und gibt diese Werte aus. Anschließend wird die Prozedur TEST aufgerufen. Laut der Parametervereinbarung wird der erste Parameter als Wertparameter, und der zweite Parameter als Variablenparameter übergeben. In der Prozedur ist die Variable c vereinbart, welcher der Wert 3 zugewiesen wird. Die nächste Ausgabe lautet also '8 19 3'.

Dann werden a und b neu besetzt. So erfolgt die Ausgabe '5 7 3'. Nach dem END-Befehl wird das Programm im Hauptprogramm fortgesetzt. Die Variable c war in der Prozedur lokal gültig, im Hauptprogramm wird ihr Wert also wieder durch den alten Wert von c ersetzt, welcher noch immer 1 ist. Die Variable b wurde als Wertparameter übergeben, und hat daher im Hauptprogramm immer noch den Wert 8. Die Variable a war hingegen ein Variablenparameter und wurde in der Prozedur

unter dem Namen b angesprochen. Sie erhielt in der Prozedur den Wert 7, und liefert ihn daher auch im Hauptprogramm. Die vollständige Ausgabe des Programms lautet also:

```
19  8 1
 8 19 3
 5  7 3
 7  8 1
```

Wird eine Variable in der Prozedur nicht gefunden, wird sie im nächst höheren Block gesucht, in unserem Beispiel also im Hauptprogramm. Die Gültigkeit wurde bereits in der Einleitung gezeigt. Vergleichen Sie die Graphik aus Kapitel 6/2.1 mit den hier gemachten Angaben. Auf diese Weise können Variablen auch global im gesamten Programm verwendet werden. Dieser 'Seiteneffekt' kann aber auch gefährlich werden, denn wenn Sie die Deklaration einer Variablen vergessen, wird ggf. eine Variable aus dem übergeordneten Block verwendet.

Ein Beispiel für den Gebrauch der sogenannten Vorwärtsdeklaration ist das folgende Programm. Es berechnet die Hilbertkurven der Ordnung eins bis sechs, und stellt diese als gemeinsame Graphik auf dem Bildschirm dar. Die Hilbertkurven der Ordnung eins bis drei zeigt Bild 6/2.7-1, und das Resultat des Programms Bild 6/2.7-2. Da sich die Hilbertkurven aus Drehungen und geeigneten Verbindungslinien der Hilbertkurve der Ordnung 1 zusammensetzen, ist für jede Richtung eine Prozedur mit Namen a,b,c bzw. d definiert worden. Diese Prozeduren rufen sich gegenseitig auf, und zeichnen so die Gesamtkurve. Die hier bereits verwendeten Graphikbefehle werden wir später noch genauer betrachten.

Wie ist es überhaupt möglich, daß sich die Prozeduren gegenseitig aufrufen? Die Bedingung für die Definition einer Prozedur besagt, daß sie vor ihrem ersten Aufruf definiert sein muß. Eine solche Anordnung ist aber hier nicht möglich. Daher werden die Prozedurköpfe bereits *vor* der eigentlichen Prozedur definiert, und mit dem reservierten Wort 'FORWARD' dem Compiler mitgeteilt, daß die Definition später erfolgt. Bei den Definitionen braucht man die Parameterliste dann nicht mehr angeben.

Für unsere drei Prozeduren lautet diese Vorwärtsdeklaration dann:

```
PROCEDURE b(i:INTEGER); FORWARD;
PROCEDURE c(i:INTEGER); FORWARD;
PROCEDURE d(i:INTEGER); FORWARD;
```

Die eigentliche Definition kann dann später erfolgen. Sie sieht für unsere Prozedur a folgendermaßen aus:

```
PROCEDURE a(i:integer);
BEGIN IF i>0 THEN BEGIN
                    d(i-1); x:=x-h; plotit;
                    a(i-1); y:=y-h; plotit;
                    a(i-1); x:=x+h; plotit;
                    b(i-1);
                    if GETKEY< >CHR(0) THEN GOTO 1;
                        (* Abbruch bei Tastendruck *)
                    END
END;
```

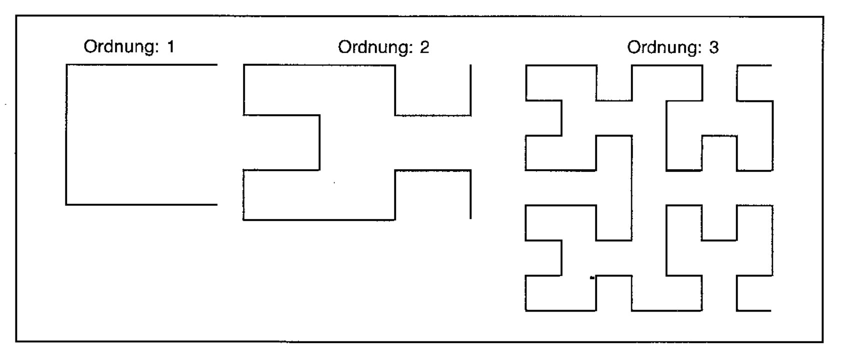


Bild 6/2.7-1 Die Hilbertkurven der Ordnung 1, 2 und 3

Hier folgt nun das vollständige Programm:

```
PROGRAM hilbert_kurve;  (* Zeichnet die Hilbert-Kurven *)

LABEL 1;         (* Label fuer Abbruch mit Taste *)

CONST n=6;       (* Hilbertkurven 1 bis 6 malen *)
      h0=256;  (* Grosse des Bildschirms 256*256 *)

VAR   i,h,                  (* Nr. der Kurve / Groesse der Kurven *)
      x,y,xm,ym,            (* Koordinaten der Linie              *)
      x0,y0:   INTEGER;     (* Startkoordinaten *)

PROCEDURE plotit;           (* Linie von alter Pos. nach x/y ziehen *)
BEGIN
  PLOT(2,ym,xm,y,x);
  xm:=x; ym:=y;
END;

(* Vorwaertsdefinition der Prozeduren b,c und d *)
PROCEDURE b(i:INTEGER); FORWARD;
PROCEDURE c(i:integer); FORWARD;
PROCEDURE d(i:integer); FORWARD;

PROCEDURE a(i:integer);
BEGIN IF i>0 THEN BEGIN
                    d(i-1); x:=x-h; plotit;
                    a(i-1); y:=y-h; plotit;
                    a(i-1); x:=x+h; plotit;
                    b(i-1);
                    if GETKEY<>CHR(0) THEN GOTO 1; (* Abbruch bei
                       Tastendruck *)
                  END
END;

PROCEDURE b;
BEGIN IF i>0 THEN BEGIN
                    c(i-1); y:=y+h; plotit;
                    b(i-1); x:=x+h; plotit;
                    b(i-1); y:=y-h; plotit;
                    a(i-1);
                  END
END;

PROCEDURE c;
BEGIN IF i>0 THEN BEGIN
                    b(i-1); x:=x+h; plotit;
                    c(i-1); y:=y+h; plotit;
                    c(i-1); x:=x-h; plotit;
                    d(i-1);
                  END
END;
```

```
PROCEDURE d;
BEGIN IF i>0 THEN BEGIN
                    a(i-1); y:=y-h; plotit;
                    d(i-1); x:=x-h; plotit;
                    d(i-1); y:=y+h; plotit;
                    c(i-1);
                  END
END;

(* Hauptprogramm *)

BEGIN
  HIRES(1); PAPER(0); INK(1);          (* Graphik einschalten *)
  PLOT(0,0,0,0,0); PLOT(1,0,0,0,0);  (* Bildschirm loeschen *)
  i:=0; h:=h0;                (* Startkoordinaten berechnen *)
  x0:=h DIV 2;
  y0:= h DIV 2;
  REPEAT                      (* Hilbertkurven zeichnen      *)
    i:=i+1;
    h:=h DIV 2;
    x0:=x0+(h DIV 2);
    y0:=y0+(h DIV 2);
    x:=x0; y:=y0;
    xm:=x; ym:=y;
    a(i)                      (* Prozeduren aufrufen         *)
  UNTIL i=n;                  (* bis alle Kurven gezeichnet *)
  WHILE GETKEY=CHR(0) DO; (* auf Tastendruck warten *)
1:HIRES(0);                     (* Graphik ausschalten      *)
END.
```

Beachten Sie bitte, daß der Bildschirm nicht das gesamte Resultat anzeigt. Das Programm erzeugt die Bilder auf einem quadratischen Bildschirm mit einer Seitenlänge, die eine Potenz von 2 sein muß. Auf dem Bildschirm des C 64 wäre also maximal die Seitenlänge 128 ($=2^7$) vollständig darstellbar. Dies ist natürlich etwas zu klein. Im Programm wurde daher die Größe 256 ($=2^6$) angegeben. Da der C 64 aber nur eine Auflösung von 320*200 Punkten besitzt, wird der obere Teil der Lösung abgeschnitten. Sollte Sie dies stören, können Sie die Konstante h0 auf den Wert 128 setzen.

Bild 6/2.7-2 Das Ergebnis des Programmlaufs.

6/2.7.2

Funktionen

Sicherlich kennen Sie aus dem BASIC des C64 die Möglichkeit, mit Hilfe des DEF FN-Befehls eine eigene Funktion zu definieren. In PASCAL haben Sie diese Möglichkeit auch. Allerdings ist hier die Funktion nicht auf eine Programmzeile beschränkt.

Funktionen sind in PASCAL eigentlich das gleiche wie Prozeduren, und werden daher gelegentlich auch als Funktionsprozeduren bezeichnet. Der Unterschied liegt nur darin, daß Funktionen einen Funktionswert zurückliefern. Sonst können sie, wie die Prozeduren, aus einem vollständigen Programmblock, eigenen Variablen- und Typvereinbarungen und eigenen Prozedur- oder Funktionsdefinitionen bestehen.

Um die Funktion von der Prozedur zu unterscheiden, wird sie durch das reservierte Wort „FUNCTION“ eingeleitet. Dann folgt, wie bei der Prozedur, der Funktionsname und gegebenenfalls die Parameterliste. Auch bei den Funktionen sind Wert- und Variablenparameter zugelassen.

Da eine Funktion einen Wert zurückgeben kann, müssen wir bei den Funktionen noch den Typ des Rückgabewertes angeben. Diese Angabe erfolgt wie bei einer Variablendefinition. Der Typename wird, durch einen Doppelpunkt getrennt, hinter die Vereinbarung (hier: hinter der Parameterliste) geschrieben.

Hier ein paar Beispiele für zulässige Funktionsköpfe:

```
FUNCTION test(a,b,c:INTEGER; VAR i:REAL) : REAL;
FUNCTION fact(n:INTEGER):INTEGER;
FUNCTION kleiner:BOOLEAN;
```

Um die Handhabung von Funktionen kennenzulernen, wollen wir ein kleines Programm schreiben, welches die ersten acht Zeilen des Pascalschen Dreiecks berechnet. Der Name „Pascalsches Dreieck“ stammt nicht etwa von der hier verwendeten Programmiersprache, sondern wieder von dem französischen Mathematiker Blaise Pascal.

Sehen wir uns zunächst die Ausgabe des Programms an:

```
0                   1
1                 1   1
2               1   2   1
3             1   3   3   1
4           1   4   6   4   1
5         1   5  10  10   5   1
6       1   6  15  20  15   6   1
7     1   7  21  35  35  21   7   1
```

Ganz links steht die Nummer der Zeile. In dem Dreieck stehen die Binominalkoeffizienten:

$$
\begin{array}{lccccccccc}
0 & & & & & \binom{0}{0} & & & & \\
1 & & & & \binom{1}{0} & & \binom{1}{1} & & & \\
2 & & & \binom{2}{0} & & \binom{2}{1} & & \binom{2}{2} & & \\
3 & & \binom{3}{0} & & \binom{3}{1} & & \binom{3}{2} & & \binom{3}{3} & \\
4 & \binom{4}{0} & & \binom{4}{1} & & \binom{4}{2} & & \binom{4}{3} & & \binom{4}{4}
\end{array}
$$

Die Binominalkoeffizienten sind folgendermaßen definiert:

$$\binom{n}{k} = \frac{n!}{k! * (n-k)!} \binom{n}{k}$$

Eine praktische Bedeutung haben die Binominalkoeffizienten im Bereich der Kombinatorik. Sie geben an, wie viele Möglichkeiten man hat, k Objekte aus n Objekten auszuwählen. So gibt es zum Beispiel 49 über 6 (= 13.983.816) Möglichkeiten, einen Block auf dem Lottoschein auszufüllen. Daher ist die Chance, 6 Richtige im Lotto zu haben, 1 zu 13.983.816.

Das Pascalsche Dreieck bietet eine einfache Möglichkeit, die Binominalkoeffizienten zu berechnen, denn es gilt:

$$\binom{n}{o} = 1 \text{ und } \binom{n}{n} = 1$$

Damit hat man die ersten zwei Zeilen bereits vollständig, und für die restlichen Zeilen die Ränder:

```
0              1
1            1   1
2          1       1
3        1           1
4      1               1
```

Die restlichen Zeilen ergeben sich nun, indem man die beiden links und rechts über der zu berechnenden Zahl stehenden Werte addiert:

```
0                 1
1               1   1
2             1   2   1             2=1+1
3           1   3   3   1           3=1+2,  3=2+1
4         1   4   6   4   1         4=1+3,  6=3+3,  4=3+1
5       1   5  10  10   5   1       5=1+4,  10=4+6,  usw.
```

Nach diesem Ausflug in die Mathematik zurück zu unserem Programm. Wir werden die Berechnungsmöglichkeit mit dem Pascalschen Dreieck nicht benutzen, sondern einfach die oben angegebene Formel verwenden. Dies hat den Nachteil, daß wir mit INTEGER-Zahlen nur bis zur achten Zeile kommen, dann wird in der Fakultätsberechnung der Zahlenbereich überschritten. Aber um Funktionen kennenzulernen, reicht dies völlig aus.

Wir beginnen unser Programm wie üblich mit der Vereinbarung des Programmnamens. Als Variablen vereinbaren wir hier *spalte* und *zeile* für die spätere Ausgabe.

```
PROGRAM pascal_dreieck;          (* berechnet die ersten 8
                               Zeilen des Pascalschen Dreiecks *)

VAR zeile,spalte:INTEGER;        (* Zeile und Spalte für die
                                    Ausgabe                    *)
```

Dann folgt die Vereinbarung der Fakultätsfunktionen *„fakultaet"*. Sie erhält einen Übergabeparameter *n* und liefert einen Wert des Typs *INTEGER*. Da wir uns jetzt noch nicht mit der Art der Berechnung beschäftigen wollen, verschieben wir die Definition mit der Anweisung *„FORWARD"* auf später.

```
(* die Fakultaetsfunktion folgt später                        *)
FUNCTION fakultaet(n:INTEGER):INTEGER; FORWARD;
```

Die Funktion *„binomi"* berechnet uns „n über k". Sie erhält dazu die Übergabeparameter *n* und *k* und liefert uns dann einen ganzzahligen Wert zurück. In der Funktion verwenden wir die lokale Hilfsvariable *„bruch"*. In ihr speichern wir das Zwischenergebnis *„k!*(n-k)!"*. Das berechnete Ergebnis wird als Funktionsergebnis zurückgeliefert, indem den Funktionsnamen *„binomi"* mit dem Zuweisungsoperator *„:="* ein Ergebnis zugewiesen wird.

Der Funktionsname verhält sich praktisch wie eine Variable. Allerdings darf er nur links von dem Zuweisungsoperator stehen, denn wenn er rechts von dem Zuweisungsoperator steht, wird er als Funktions*aufruf* behandelt.

```
(* Berechnen der Binomialkoeffizienten n über k              *)
FUNCTION binomi(n,k:INTEGER):INTEGER;
VAR bruch:INTEGER;               (* Hilfsvariable             *)
BEGIN
  bruch:=fakultaet(k)*FAKULTAET(n-k); (* bruch= k!*(n-k)! *)
  binomi:=fakultaet(n) DIV bruch;     (* Ergebnis=n!/bruch*)
END;
```

Jetzt wollen wir die Definition der Fakultätsfunktion *„fakultaet"* nachholen. Wir haben die Fakultät einer Zahl schon auf viele Arten programmiert. Hier wollen wir eine weitere Variante kennenlernen. Sie wissen, daß die Fakultät einer Zahl n als

```
n*(n-1)*(n-2)*(n-3)* . . . *1
```

berechnet wird, und daß *0!* eins ergibt. Wenn Sie sich die Formel für *n* genau ansehen, erkennen Sie, daß

```
n!=n*(n-1)!
```

ist. Von dieser Tatsache machen wir hier Gebrauch:

```
(* jetzt die Definition von fakultaet                          *)
FUNCTION fakultaet;
BEGIN
  IF n=0 THEN fakultaet:=1                  (* 0! = 1          *)
  ELSE fakultaet:=n*fakultaet(n-1);         (* n! = n*(n-1)!   *)
END;
```

In der Fakultätsfunktion testen wir, ob *n* Null war. Dann liefern wir als Funktionsergebnis die 1. Andernfalls wenden wir unsere Regel an und schreiben

```
fakultaet:=n*fakultaet(n-1)
```

Die Funktion ruft sich an dieser Stelle selbst auf. Da dabei *n* um 1 vermindert wird, ist sichergestellt, daß die Funktion irgendwann einmal mit der Berechnung von 0! beendet wird. Diese Technik wird *„Rekursion"* genannt.

Was uns nun noch fehlt, ist das Hauptprogramm. In ihm lassen wir eine Schleife *„zeile"* über die Zeilen laufen und darin eine Schleife *„spalte"* über die Spalten. Was bleibt, ist die Ausgabe der Zeilennummer *„zeile"* an der linken Seite und das Einrücken auf die erste Position. Letzeres erfolgt durch die Ausgabe eines Leerzeichens in einem Feld der Länge *15-zeile*2*. Diese Ausgabe kommt einem Tabulator-Befehl gleich.

```
BEGIN                               (* Hauptprogramm          *)
  FOR zeile:=0 TO 7 DO              (* 7 Zeilen berechnen     *)
  BEGIN
    WRITE(zeile:3,' ':15-zeile*2);  (* Nummer ausgeben und
                                       zur ersten Position *)
    FOR spalte:=0 TO zeile DO       (* Anzahl Spalten =
                                       Nummer der Zeile +1 *)
      WRITE(binomi(zeile,spalte):4);(* Wert ausgeben        *)
    WRITELN;                         (* neue Zeile          *)
  END;
END.
```

Merke: Eine Funktionsdefinition beginnt mit dem reservierten Wort *FUNCTION*.

Der Parameterliste muß der Typ der Funktion, durch einen Doppelpunkt getrennt, folgen.

Dem Funktionsnamen kann innerhalb der Funktionsdefinition ein Wert durch den Zuweisungsoperator „:=“ zugewiesen werden. Dieser Wert ist der Rückgabewert der Funktion.

Steht der Funktionsname rechts von dem Zuweisungsoperator „:=“, gilt dies als Funktionsaufruf.

6/2.8

Strukturierte Datentypen

PASCAL bietet eine ganze Reihe von Datentypen. So haben wir bereits die Typen CHAR, INTEGER, REAL und BOOLEAN als Grundtypen kennengelernt. Darüber hinaus konnten wir eigene Typen in Form von Aufzählungen definieren und Felder von Variablen vereinbaren.

In den folgenden Kapiteln werden wir darüber hinausgehen. Wir werden Variablen zu Verbunden zusammenfassen, die Dateiverwaltung besprechen, den Typ der Menge kennenlernen und uns mit Zeigern beschäftigen.

6/2.8.1

Verbunde

Beginnen wir mit den Verbunden. Ein natürlicher Verbund ist die Adresse eines Menschen. Sie besteht aus den Komponenten Straße, Hausnummer, Postleitzahl und Wohnort. Wir wollen hier noch den Namen und den Vornamen hinzuzählen.

All diese Komponenten fassen wir unter dem Begriff *Adresse* zusammen. PASCAL bietet uns die Möglichkeit, dies in der Programmiersprache nachzubilden. Mit unseren bisherigen Kenntnissen müßten wir die Komponenten einzeln vereinbaren:

```
VAR name, vorname  :  PACKED ARRAY [1..20] OF CHAR;
    strasse, ort   :  PACKED ARRAY [1..20] OF CHAR;
    plz, haus      :  INTEGER;
```

Wollen wir dies unter dem Namen *Adresse* zusammenfassen, benutzen wir die *RECORD*-Anweisung:

```
TYPE string  =  PACKED ARRAY [1..20] OF CHAR;

     adresse =  RECORD
                  name, vorname  :  string;
                  strasse, ort   :  string;
                  plz, haus      :  INTEGER;
                END;

VAR Name1, Name2 : adresse;
```

Wir haben uns einen Typ mit dem Namen *adresse* definiert, welcher aus den Komponeneten *name, vorname, strasse, ort, plz* und *haus* besteht. Das *abschließende END* beendet den Verbund.

In der nachfolgenden *VAR*-Anweisung werden zwei Variablen *name1* und *name2* definiert, die von der Struktur des Typs *adresse* sind. Wir hätten das gleiche mit der Anweisung

```
VAR name1, name2 : RECORD
                     name, vorname  :  string;
                     strasse, ort   :  string;
                     plz, haus      :  INTEGER;
                   END;
```

erreichen können. Dann hätten wir für das folgende Programm aber den Typ *adresse* nicht zur Verfügung. Vielleicht brauchen wir ihn ja noch einmal.

Nehmen wir nun an, Sie wollen ein Feld für 100 Personen anlegen. In Sprachen wie BASIC müßten Sie dann für jede Komponente ein Feld anlegen und verwalten. Anders in PASCAL:

```
VAR personen : ARRAY [1..100] OF adresse;
```

Variablen des Types *„adresse"* lassen sich wie alle anderen Variablen benutzen. Eine Zuweisung der Form

```
name1 := name2;
```

bewirkt daher, daß alle Komponenten von *„name1"* auf die Werte der Komponenten von *„name2"* gesetzt werden.

Merke: Verbunde werden wie normale Variablen behandelt.

Was uns noch fehlt, ist eine Möglichkeit, einzelne Komponenten eines Verbundes anzusprechen. Aber auch dies ist ganz einfach. Wir brauchen nur hinter den Variablennamen den Komponentennamen, durch einen Punkt getrennt, zu schreiben. Dann bezeichnet

name1.name	den Familiennamen der Person in *name1*
name1.vorname	den Vornamen der Person in *name1*
name1.strasse	die Straße der Person in *name1*
name1.ort	den Wohnort der person in *name1*
name1.plz	die Postleitzahl des Wohnortes der Person in *name1*
name1.haus	die Hausnummer der Person in *name1*

Diese Ausdrücke repräsentieren Variablen. Man kann ihnen daher Werte zuweisen oder Werte aus ihnen lesen.

Merke: Die Komponenten eines Verbundes können angesprochen werden, indem hinter den Variablennamen des Verbundes der Komponentennamen, durch einen Punkt getrennt, geschrieben wird:

<Verbund_Name>.<Komponenten_Name>

Probieren Sie nun das folgende kleine Beispiel aus. Es liest die Daten von fünf Personen ein und ermittelt dann, welche Person mit welcher Person in einem Haus lebt:

```
PROGRAM person_vergleich;

CONST pers_zahl = 5;          (* Anzahl der Personen      *)
```

```
TYPE string = PACKED ARRAY[1..10] OF CHAR; (* String def. *)

     adresse = RECORD              (* Verbund fuer "adresse"    *)
                 name, vorname: string;
                 strasse, ort : string;
                 plz, haus    : INTEGER;
               END;

VAR person: ARRAY[1..pers_zahl] OF adresse; (* Feld der
                                              Personen      *)
    p1,p2: adresse;              (* Zwei Variablen fuer den
                                    Vergleich               *)
    i,j:   INTEGER;              (* Laufvariablen            *)

(* Einlesen einer Adresse                                    *)
PROCEDURE einlesen(VAR name:adresse);
BEGIN
  WRITELN;                       (* neue Zeile               *)
  WRITE('Vorname:    '); READLN(name.vorname); (* Vorname    *)
  WRITE('Name:       '); READLN(name.name);    (* Name       *)
  WRITE('Strasse:    '); READLN(name.strasse); (* Strasse    *)
  WRITE('Hausnummer:'); READLN(name.haus);     (* Nummer     *)
  WRITE('Wohnort:    '); READLN(name.ort);     (* Ort        *)
  WRITE('Plz.:       '); READLN(name.plz);     (* Plz        *)
END; (* einlesen *)

BEGIN
  FOR i:=1 TO pers_zahl DO       (* alle Personen einlesen   *)
    einlesen(person[i]);

  FOR i:=1 TO pers_zahl-1 DO     (* jede Person ...          *)
    FOR j:=i+1 TO pers_zahl DO(* ... mit jeder testen        *)
      BEGIN
        p1:=person[i];           (* p1 und p2 sind die       *)
        p2:=person[j];           (*   Personen               *)

        IF (p1.ort=p2.ort) AND(* Ort gleich ?                *)
           (p1.plz=p2.plz) AND(* und Plz gleich ?            *)
           (p1.strasse=p2.strasse) AND(* u.Strasse gleich?*)
           (p1.haus=p2.haus) THEN (* und Haus gleich?        *)

           BEGIN                 (* ja, dann Namen ausgeben *)
             WRITE  (p1.vorname,' ',p1.name);
             WRITE  (' lebt bei ');
             WRITELN(p2.vorname,' ',p2.name);
           END; (* Then *)

      END; (* For j *)
END.
```

Betrachten wir noch einmal die Prozedur *einlesen*. Leider ist in PASCAL nicht vorgesehen, Verbunde einzulesen. Wir müssen daher die Komponenten einzeln einlesen. Dies ist aber nicht unbedingt ein Nachteil von PASCAL, denn ohne die Angabe, welche Komponente gerade eingelesen werden soll, ist der Benutzer sowieso ziemlich ratlos. Und für diese Angabe benötigen wir nun einmal vorangehende *WRITE*-Befehle.

Aber an einer anderen Stelle erleichtert uns PASCAL die Arbeit. In der Prozedur *einlesen* bezogen sich alle Komponentenangaben auf die Variable *name*. PASCAL bietet für solche Fälle die Möglichkeit, dies dem Compiler mitzuteilen. Man erreicht dies durch die Anweisung

```
WITH <var_name> DO
```

wobei *<var_name>* hier für den gewünschten Variablennamen steht. Ändern wir also unsere Prozedur *einlesen* also etwas ab:

```
PROCEDURE einlesen(VAR name:adresse);
BEGIN
  WRITELN;
  WITH name DO BEGIN
    WRITE('Vorname:    '); READLN(vorname);
    WRITE('Name:       '); READLN(name);
    WRITE('Strasse:    '); READLN(strasse);
    WRITE('Hausnummer:'); READLN(haus);
    WRITE('Wohnort:    '); READLN(ort);
    WRITE('Plz.:       '); READLN(plz);
  END; (* With *)
END; (* einlesen *)
```

Wie Sie sehen, konnten wir uns 6 mal die Angabe *„name“* sparen.

Merke: Mit der *WITH . . . DO*-Anweisung kann ein Abschnitt festgelegt werden, in dem sich alle Komponentennamen auf *einen* Verbund beziehen.

Manchmal ist es sinnvoll, einen Verbund in verschiedenen Variationen zu definieren. Wenn wir bei unserem Beispiel der Personendaten bleiben, kann es zum Beispiel sinnvoll sein, je nach Geschlecht oder Familienstand verschiedene weitere Daten hinzuzunehmen. Mit unseren bisherigen Kenntnissen haben wir zwei Möglichkeiten, dies zu realisieren:

Die erste Möglichkeit besteht darin, für jeden Fall einen eigenen Verbund zu definieren. Dann müßten wir aber auch bei den Feldern für jeden Verbund ein entsprechendes Feld definieren, was zu einer großen Speicherplatzverschwendung führt.

Als zweite Möglichkeit könnten wir alle möglichen Komponenten in einen Verbund definieren und dann nur die Komponenten benutzen, welche für unseren Fall gerade von Bedeutung sind. Diese Methode kann man vertreten, wenn wesentlich mehr gleiche Komponenten als verschiedene Komponenten in dem Verbund vorkommen. Sonst vergeuden wir auch hier Speicherplatz.

PASCAL bietet uns hier eine interessante Alternative: den varianten Verbund. Bei dem varianten Verbund hat man die Möglichkeit, eine Variable anzugeben, welche bestimmt, wie der Rest des Verbundes aufgebaut ist. Dies sehen wir uns am besten an einem Beispiel an. In dem folgenden Programm ist der Verbund *adresse* als varianter Verbund aufgebaut:

```
PROGRAM personen;

CONST pers_zahl = 5;

TYPE string = PACKED ARRAY[1..10] OF CHAR;
     t_geschl=(maennlich,weiblich);

     adresse = RECORD
                  name, vorname: string;
                  strasse, ort : string;
                  plz, haus    : INTEGER;
                  CASE geschlecht:t_geschl OF
                    maennlich:(alter:INTEGER;
                               baertig:0..1);
                    weiblich: (verheiratet:0..1;
                               haarfarbe:INTEGER)
                END;
VAR person: ARRAY[1..pers_zahl] OF adresse;
    i:INTEGER;
```

Die Komponenten *name, vorname, plz, ort, strasse* und *haus* sind geblieben. Sie gehören zu den gemeinsamen Komponenten. Daher hat sich bisher am Programmtext nichts geändert. Dann folgt die Anweisung

```
CASE geschlecht : t_geschl OF
```

Mit dieser Anweisung wird der variante Teil des Verbundes eingeleitet. Die Variabe *geschlecht* wird als weitere *gemeinsame* Komponente in den Verbund aufgenommen. Da sie vom Typ *t_geschl* ist, kann sie nur die Werte *männlich* oder *weiblich* annehmen. Dieser Wert entscheidet nun über den weiteren Aufbau des Verbundes.

Hat die Variable *geschlecht* den Wert *männlich*, so sollen in dem Verbund noch die Komponenten *alter* und *baertig* aufgenommen werden. Diese Komponenten werden in der ersten Klammer vereinbart.

Hat die Variable *geschlecht* hingegen den Wert *weiblich*, so sollen in dem Verbund noch die Komponenten *verheiratet* und *haarfarbe* aufgenommen werden. Diese Komponenten werden in der zweiten Klammer vereinbart.

Könnte die Variable *geschlecht* noch weitere Werte annehmen, so müßte man für diese Werte weitere Listen in der CASE-Anweisung aufnehmen.

Mit dem Zugriff auf die Komponenten muß man als Programmierer aber sehr vorsichtig sein. Zumindest bei Oxford-PASCAL ist es möglich, auf *alle* Komponenten zuzugreifen, unabhängig welchen Aufbau der Verbund gerade hat. Sie müssen daher in Ihrem Programm sicherstellen, daß Sie nur auf Komponenten zugreifen, die auch vorhanden sind. Betrachten wir dies am Beispiel der Prozedur *einlesen*.

```
PROCEDURE einlesen(VAR name:adresse);
VAR gesch: 0..1;
BEGIN
  WRITELN;
  WITH name DO BEGIN
    WRITE('Vorname:    '); READLN(vorname);
    WRITE('Name:       '); READLN(name);
    WRITE('Strasse:    '); READLN(strasse);
    WRITE('Hausnummer:'); READLN(haus);
    WRITE('Wohnort:    '); READLN(ort);
    WRITE('Plz.:       '); READLN(plz);
    WRITE('Geschl.0/1  '); READLN(gesch);
    IF gesch=0 THEN geschlecht:=weiblich
               ELSE geschlecht:=maennlich;
    CASE geschlecht OF
    maennlich:BEGIN
```

```
                    WRITE('Alter:      '); READLN(alter);
                    WRITE('Baertig?    '); READLN(baertig);
                 END;
     weiblich :BEGIN
                    WRITE('Verheiratet'); READLN(verheiratet);
                    WRITE('Haarfarbe: '); READLN(haarfarbe);
                 END;
     END;
   END;
 END;
```

Das erste Problem begegnet uns beim Einlesen des Geschlechts. PASCAL kann keine selbst definierten Typen einlesen. Der Einfachheit halber lesen wir in diesem Beispiel nur 0 oder 1 ein und setzen mit dem nachfolgenden *IF-THEN-ELSE*-Konstrukt die Variable *„geschlecht“* auf den richtigen Wert.

Dieses Problem hat mit der Verwendung eines Verbundes natürlich nichts zu tun, sondern ist ein allgemeines Problem bei selbstdefinierten Typen.

Den Zugriff auf die varianten Komponenten erledigt man am besten in einer *CASE*-Konstruktion, welche den Aufbau des varianten Verbundes widerspiegelt. Je nach Geschlecht greifen wir auf die Komponenten *„alter“* und *„baertig“* oder auf die Komponenten *„verheiratet“* und *„haarfarbe“* zu.

Merke: Der variante Verbund hat den allgemeinen Aufbau:

RECORD
<Variablenliste> : <Datentyp>;
<Variablenliste> : <Datentyp>;
: : :
<Variablenliste> : <Datentyp>;
CASE **<Variable>: <Datentyp>** ***OF***
<Konstantenliste> : (<Komponentenliste>);
<Konstantenliste> : (<Komponentenliste>);
: : :
<Konstantenliste> : (<Komponentenliste>)
END;

6/3 FORTH

(Autor: Rainer König)

FORTH ist mehr als eine Programmiersprache, der Ausdruck „Programmierumgebung“ trifft hier wesentlich besser, da FORTH eigentlich eine Synthese aus Programmiersprache, Betriebssystem und Testhilfen ist.

FORTH wurde in den frühen 70er-Jahren von dem Amerikaner Charles Moore entwickelt, er wollte eine Sprache schaffen, die schnell und einfach zu handhaben ist. Moore setzte FORTH auf einer PDP-11 zur Steuerung von astronomischen Teleskopen ein. Das Betriebssystem dieses Rechners konnte jedoch nur Dateinamen mit bis zu 5 Zeichen verarbeiten, aus diesem Grund entstand aus dem ursprünglichen beabsichtigten Namen „FOURTH“ (Sprache der vierten Generation) die Kurzform FORTH. Auch heute wird FORTH vorwiegend zu Steuerungsaufgaben eingesetzt, sogar in der Weltraumfahrt. Aber auch andere Aufgaben sind mit FORTH zu bewältigen.

Um nun Ihr Interesse an dieser Sprache restlos zu wecken, wollen wir einen kleinen Fragebogen bringen.

— Können Sie sich eine Programmiersprache vorstellen, in der Sie Ihre Anweisungen ab einem gewissen Level fast schon in Umgangssprache an den Computer formulieren können? Ein Beispiel wäre die Belichtungszeitsteuerung im Fotolabor, hier wäre es doch toll, wenn der Computer den Satz „Lampe an, 10 sec warten, Lampe aus“ verstehen könnte.

— Können Sie sich eine Programmiersprache vorstellen, in der Sie sowohl direkt auf Maschinenebene aber auch in Hochsprachenebene programmieren können, und das in allen Fällen strukturiert, auch in der Maschinenebene?

— Können sie sich eine Programmiersprache vorstellen, in der Sie den Compiler, also den Umfang der Sprache selbst erweitern können?

— Können Sie sich eine Programmiersprache vorstellen, in der sich die Entwicklungszeit für Ihre Programmierprobleme drastisch reduziert?

Wenn Ihre Antwort auf eine dieser Fragen „Nein“ lautet, dann sollten Sie unbedingt weiterlesen, denn FORTH ist diese Sprache.

6/3.1

Erste Schritte in FORTH

So, nun sollen die ersten Schritte in FORTH gemacht werden, also schalten Sie am besten Ihren Computer an und starten Sie Ihr FORTH-System, denn am einfachsten ist es, wenn man die Beispiele sofort nachvollziehen kann. Unsere Beispiele verwenden das C-64-FORTH von Performance Micro Productions, da FORTH aber eine sehr gut standardisierte Sprache ist, lassen sich die Beispiele auch auf jedem anderen FORTH-System nachvollziehen. Aber seien Sie gewarnt: FORTH ist anders, es zeigt nur sehr wenig Verwandtschaft mit anderen Programmiersprachen. Und vor allem: Wenn einen das FORTH-Fieber erst einmal gepackt hat, dann vergißt man solche Sprachen wie BASIC nur allzugern, FORTH macht süchtig. Aber Scherz beiseite, inzwischen dürfte Ihr FORTH-Compiler geladen und gestartet sein, und er meldet sich mit einem simplen „OK“. Das erinnert irgendwie an BASIC, hier meint der Computer eben „READY“, Forth ist eine weniger „fertige“ Sprache, in FORTH ist eben alles „OK“. Um nur mal so nebenbei den Sprachumfang aufzuzeigen, geben wir gleich mal das Wort VLIST ein und drücken die RETURN-Taste. Hoffentlich fallen Sie jetzt nicht vor Schreck vom Stuhl, denn das, was jetzt alles auf dem Bildschirm vom Computer erscheint ist der Grundwortschatz von FORTH, es dürften so um die 250-400 Worte sein. Doch keine Panik! Immerhin konnte man ja schon einige bekannte Wörter ausmachen, z.B. „+“, „—“ und die anderen Rechensymbole, also wird FORTH doch hoffentlich rechnen können . . .

Somit wollen wir uns nun gleich mit einigen Rechenbeispielen befassen. Hier erleben wir nun auch sofort die erste Besonderheit von FORTH, es rechnet nämlich mit UPN. Was ist denn das???

UPN ist die Kurzform für „Umgekehrt polnische Notation“; keine Angst, Sie müssen der polnischen Sprache nicht mächtig sein, um UPN zu verstehen, der Name kommt nur daher, daß ein polnischer Mathematiker als erster die Idee hatte. Um Rechnungen an den Computer zu formulieren, kann man ja verschiedene Schreibweisen verwenden, z.B. 2+3, hier wird also erst der erste Operand (2) angegeben, dann die Operation (+) und schließlich der zweite Operand (3). Diese Form ist in den meisten Programmiersprachen üblich. UPN heißt nun nichts anderes, als daß man zuerst alle Operanden angibt, und dann die Operation, die damit auszuführen ist. Die Taschenrechner der Firma HP arbeiten beispielsweise mit dieser Technik.

Im gewählten Beispiel müßte man also 2 3 + schreiben. Geben wir es doch mal in FORTH ein, so erscheint das gewohnte OK, es sei denn, Sie haben die Leerzeichen

zwischen den Operanden und dem Operator nicht eingegeben. Das Leerzeichen ist in FORTH nämlich sehr wichtig, es trennt die einzelnen **Worte** im **Befehlsstrom.**

OK, ganz gut, aber wo ist das Ergebnis? Machen wir es nicht so spannend, geben Sie einfach den Dezimalpunkt ein (.), und FORTH liefert das korrekte Ergebnis, nämlich 5.

Was ist nun eigentlich passiert? Durch die Angabe der Zahl 2 wurde FORTH veranlaßt, diese Zahl auf dem Datenstack abzulegen. Der Datenstack ist ein Speicherbereich, der wie ein Stapel funktioniert. Man kann Informationen hereinschreiben und herausholen, aber nur nach einem System, dem LIFO-System (Last In First Out). Das bedeutet nur, daß man den letzten hineingeschriebenen Wert als ersten wieder ausliest. Die ganze Stapelei läßt sich ganz einfach mit einem Kartenstapel vergleichen, auf den man nur oben Karten drauflegen kann, oder eben von oben Karten wegnehmen kann. In unserem Beispiel haben wir eben eine Karte mit der Zahl 2 abgelegt. Im nächsten Schritt geschieht dasselbe, nur eben mit der Zahl 3. Nun stehen also schon zwei Zahlen auf dem Stack, zuoberst die 3. Das + bewirkt nun, daß die beiden obersten Werte vom Stack geholt werden, dann werden sie addiert und das Ergebnis wird auf den Stack gelegt. Die beiden Operanden sind dann allerdings verloren. Mit . wird dann das oberste Stackelement auf dem Bildschirm ausgegeben, und dadurch auch vom Stack entfernt. Nun können wir ein paar Rechenbeispiele nachvollziehen:

25 4 — . ergibt die Ausgabe von 21.
5 6 * . ergibt als Ergebnis 30.
10 3 / . ergibt als Ergebnis den Wert 3.

Hoppla, bei der letzten Rechnung hätte eigentlich 3.33333333 herauskommen sollen! Hier also die zweite Eigenheit von FORTH, es kann vorerst nur mit Integer-Zahlen (ganze Zahlen im Bereich von —32768..32767 operieren). Aber das ist kein Nachteil, wie wir später sehen werden.

Die UPN-Schreibweise bei FORTH bedingt zudem, daß Klammern in Rechenausdrücken entfallen können. Auch hier ein Beispiel: (2+3) * (4+5) ist in FORTH folgendermaßen zu formulieren:

2 3 + 4 5 + *

Ist doch ganz einfach, oder? Zuerst werden die beiden Klammerausdrücke berechnet und dann folgt die Multiplikation. Im Klartext: Zuerst werden die Zahlen 2 und 3 auf dem Stack abgelegt und addiert (+). Nun legen wir die Zahlen 4 und 5 oben auf den Stack und addieren auch sie (+). Auf dem Stack befinden sich nun die Ergebnisse beider Additionen, diese werden schließlich mit * multipliziert, und das Endergebnis bleibt oben auf dem Stack liegen.

Wer das nun als umständlich bezeichnet, sollte folgendes wissen:

— Die angesehene Firma Hewlett Packard stellt ausschließlich Taschenrechner her, die auf diesem System basieren, da man sich dadurch effektiv Arbeit sparen kann.

— Hochsprachencompiler übersetzen Programme häufig in einen Zwischencode, der sehr nahe an FORTH herankommt, und bei Arithmetik genau dieses System verwendet. Ein Paradebeispiel hierfür ist das weltbekannte UCSD-Pascal (UCSD=University of California at San Diego). Der von UCSD-Pascal erzeugte sogenannte P-Code ist fast schon FORTH.

6/3.2

Ein erstes Programm

Nun wollen wir aber mal voll einsteigen und ein kleines Programm in FORTH schreiben. Auch hier hat FORTH wenig Ähnlichkeit mit konventionellen Programmiersprachen, ein FORTH-Programm zu schreiben, heißt nichts anderes, als den Wortschatz des FORTH-Systems (mit VLIST auflistbar) zu erweitern. Wählen wir als Beispiel eine Umrechnung von Dezimalzahlen nach Hexadezimalzahlen, eine Aufgabenstellung, die in FORTH sehr elegant zu lösen ist. Eine mögliche Lösung sieht so aus:

```
: D—> HDUP DECIMAL ." DEZIMAL " . . " = HEXADEZIMAL " HEX . DECIMAL ;
```

Was haben wir nun im einzelnen gemacht? Der Doppelpunkt leitet die Definition eines neuen Wortes ein, in FORTH-Kreisen nennt man so etwas „Colon-Definition". Nun folgt der Name des neu zu definierenden Wortes, in unserem Falle ist dies „D—>H". Es folgt die Kette von FORTH-Wörtern, welche beim Aufruf des neuen Wortes ausgeführt werden sollen. In unserem Fall wird das oberste Stack-Element dupliziert (DUP d.h., das oberste Stackelement ist jetzt zweimal vorhanden), sodann wird es im dezimalen Zahlensystem ausgegeben. Nun wird FORTH auf das Hexadezimalsystem eingestellt (HEX), und das (durch das Duplizieren zweimal vorhandene) oberste Element in diesem Zahlensystem ausgegeben. Schließlich wird FORTH wieder auf das Dezimalsystem gesetzt. Der Strichpunkt beendet die Colon-Definition.

Jetzt ist das neue Wort bereits ausführbar, außerdem findet man es bei der Ausführung des VLIST-Befehls an erster Stelle im Wortschatz.

In FORTH sollte man sich zur Dokumentation bei neuen Wörtern Stack-Beeinflussungen aufschreiben. Das heißt, man schreibt die Wortdefinition und die Änderung des Stacks dazu. Folgende Schreibweise hat sich dafür eingebürgert:

```
wort ( a/b/c — d/e ) Kommentar
```

Die Buchstaben „a", „b" und „c" bezeichnen den Stackinhalt vor der Ausführung des Wortes, die Buchstaben „d" und „e" den Stackinhalt nach Ausführung des Wortes. Der am weitesten rechts stehende Buchstabe kennzeichnet hier das oberste Stackelement, also bei unserem Beispiel vor der Ausführung das „c", nach der Ausführung das „e". Schließlich kann ein Kommentar folgen, der beschreibt, was das

Wort bewirkt. In unserem Umrechnungsbeispiel könnten wir also folgendes schreiben:

```
D—> H ( n — ) Gibt n in Dezimal und Hex aus
```

Schließlich sollten wir noch testen, ob unser neues Wort auch seinen Zweck erfüllt. Geben wir also nach erfolgter Definition einmal 194 D—>H ein, so sollte als Antwortzeile folgendes erscheinen:

```
DEZIMAL 194 = HEXADEZIMAL C2
```

Das Wort D—>H ist nun fester Bestandteil unseres FORTH-Vokabulars. Es kann jederzeit auch in anderen Worten verwendet werden. Wollen wir zum Beispiel eine Umrechnungstabelle drucken, in der die Umrechnungen für die Dezimalwerte von 0 bis 255 aufgeführt sind, so ist folgendes zu schreiben:

```
: TABELLE CR ." DEZ—HEX KONVERTIERUNG " CR 256 0 DO I D—>H CR LOOP ;
```

Ein paar neue FORTH-Wörter tauchen hier auf. CR gibt einen Zeilenvorschub aus. DO leitet eine Schleife ein, ähnlich der FOR-NEXT-Schleife in BASIC. Start für die Schleife ist der Wert 0, wenn der Wert 256 erreicht ist (nicht überschritten! wie in BASIC), dann wird die Schleifenausführung abgebrochen. LOOP ist das NEXT-Äquivalent in FORTH. I holt einfach den Schleifenindex auf den Stack. Die Eingabe von TABELLE löst somit den Druck einer Umrechnungstabelle aus.

Folgende Erkenntnisse über FORTH können wir nun aus den Beispielen ziehen:

— FORTH-Programmierung ist die gut durchdachte Erweiterung des Grundwortschatzes von FORTH.

— FORTH-Wörter haben den Status von „Unterprogrammen“, h.h. sie können von anderen Wörtern aufgerufen werden.

— Da nur bereits existierende Wörter aufgerufen werden, ist der Programmierstil in FORTH eine klassische „Bottom-Up“-Technik, d.h. zuerst müssen die Wörter für ganz einfache Aktionen definiert werden (als Unterprogramme der untersten Ebene), dann die entsprechenden Aufrufer und so weiter.

— Dieser Programmierstil führt zu einer guten Strukturierung der Programmieraufgabe und vor allem zu einem sehr kompakten Programm.

— Da neu definierte Wörter jederzeit austestbar sind, ist die Möglichkeit, daß sich Fehler einschleichen, relativ minimal.

Somit haben wir einen Punkt erreicht, an dem es sinnvoll erscheint, den Grundwortschatz von FORTH in der Kurzschreibweise (mit Stackinformation) anzugeben.

6/3.3

FORTH Grundwortschatz

Operanden:	n, n1 . . .	16-bit-Integers (mit Vorzeichen)
	d, d1 . . .	32-bit-Integers (mit Vorzeichen)
	u, u1 . . .	16-bit-Integers (unsigned: ohne Vorzeichen)
	ud . . .	32-bit-Integers (unsigned: ohne Vorzeichen)
	addr	16-bit-Adresse
	b	8-bit-Byte
	c	7-bit-ASCII (Character)
	f	Boolsches Flag (< >0 entspricht „wahr“)

TOS (Top of Stack) bezeichnet das oberste Stack-Element, **SEC** das zweite Stackelement.

Stack-Manipulation

-DUP	(n - n/?)	Dupliziert TOS nur dann, wenn TOS ungleich Null
>R	(n -)	Legt TOS auf Return-Stack ab
DROP	(n -)	Entfernt oberstes Stackelement
DUP	(n - n/n)	Dupliziert oberstes Stackelement
OVER	(n1/n2 - n1/n2/n1)	Legt SEC oben auf den Stack
R	(- n)	Kopiert den Return-Stack zum TOS, Return-Stack unverändert!
R>	(- n)	Holt TOS vom Return-Stack
ROT	(n1/n2/n3 - n2/n3/n1)	Rotiert die oberen drei Elemente
SWAP	(n1/n2 - n2/n1)	Vertauscht die beiden oberen Elemente des Stack

Zahlensysteme

BASE	(-addr)	Legt Adresse der USER-Variablen BASE auf TOS ab
DECIMAL	(-)	Setzt BASE auf 10 (Dezimalsystem)
HEX	(-)	Setzt BASE auf 16 (Hexadezimalsystem)

Arithmetik

*	(n1/n2 - Produkt)	16-Bit-Multiplikation (Ergebnis: 16-bit)
*/	(n1/n2/n3 - Quotient)	Wie */MOD, jedoch ohne Berechnung des Rests
*/MOD	(n1/n2/n3 - Rest Quot)	Multiplikation mit anschließender Division (n1 * n2/n3)
+	(n1/n2 - Summe)	Addiert TOS und SEC zu neuem TOS (16-bit)
-	(n1/n2 — Differenz)	Subtraktion (16-bit)
/	(n1/n2 - Quotient)	16-Bit-Ganzzahl-Division (Ergebnis: 16-bit)
/MOD	(n1/n2 - Rest Quot)	Berechnet Quotient und Rest
1+	(n - n+1)	Addiert 1 zum TOS
2+	(n - n+2)	Addiert 2 zum TOS
ABS	(n - u)	Bildet Absolutwert von TOS
D+	(d1/d2 - Summe)	Addition (32-bit)
DABS	(d - ud)	Bildet Absolutwert von 32-Bit-Zahl
DMINUS	(d - - d)	Negiert 32-bit-Zahl
M/MOD	(ud1/u2 - u3/ud4)	Division einer 32-bit-Zahl mit Übergabe Rest/Quot.
MAX	(n1/n2 - Maximum)	Beläßt Maximum von n1 und n2 auf Stack
MIN	(n1/n2 - Minimum)	Beläßt Minimum von n1 und n2 auf Stack
MINUS	(n - - n)	Negiert TOS
MOD	(n1/n2 - Rest)	Berechnet den Divisionsrest von n1/n2

Boolsche Operationen

AND	(n1/n2 - logisch UND)	Logische UND-Verknüpfung von TOS und SEC
OR	(n1/n2 - logisch ODER)	Logische ODER-Verknüpfung von TOS und SEC
XOR	(n1/n2 - logisch XOR)	Exclusiv-ODER-Verknüpfung von TOS und SEC

Vergleichsoperationen

0<	(n - f)	„wahr", wenn n<0
0=	(n - f)	„wahr", wenn n=0
<	(n1/n2 - f)	„wahr", wenn n1<n2
=	(n1/n2 - f)	„wahr", wenn n1=n2
>	(n1/n2 - f)	„wahr", wenn n1>n2

Speicherbezogene Befehle

@	(addr - n)	Holt den 16-Bit-Inhalt von Adresse
!	(n addr -)	Speichert 16-Bit-Zahl ab Adresse ab
BLANKS	(n1/n2 -)	Belegt n2 Bytes ab Adresse n1 mit Blanks (CHR$(32))
C@	(addr - b)	Holt den 8-Bit-Inhalt einer Speicherzelle (PEEK)
C!	(b addr -)	Speichert 8-Bit-Zahl in Speicherzelle ab (POKE)
CMOVE	(n1/n2/n3 -)	Überträgt n3 Zeichen von Adresse n1 nach Adresse n2
ERASE	(n1/n2 -)	Belegt n2 Bytes ab Adresse n1 mit Null
FILL	(n1/n2/b -)	Füllt n2 Bytes ab Adresse n1 mit Wert b
SP@	(- addr)	Holt Position des Stack-Pointers auf TOS
TOGGLE	(n1/n2 -)	Das Byte an Adresse n1 wird mit n2 XOR-verknüpft

Strukturworte

BEGIN . . . UNTIL	(f)	Führt Schleife so lange aus, bis f=„wahr“
BEGIN WHILE REPEAT	(f)	Wenn f=„wahr“, dann wird Schleife ausgeführt
BEGIN . . .AGAIN	(-)	Endlosschleife zwischen BEGIN und AGAIN
DO . . . LOOP	(n1 n2 -)	Initialisiert Schleife von n2 bis n1 mit Increment 1
DO . . +LOOP	(n1 n2 -)	Wie DO LOOP, jedoch mit Increment von TOS
I	(- n)	Holt Schleifenindex auf TOS
IF . . ELSE . . ENDIF	(f)	Strukturierte IF-Abfrage
LEAVE	(-)	Verläßt Schleife beim nächsten Loop

Ein-Ausgabe-Operationen

.	(n -)	Druckt TOS aus
. “	(-)	Gibt Txt bis zum nächsten “ aus
.R	(n1/n2 -)	Druckt n1 mit n2 Stellen Feldweite aus
?	(addr -)	Druckt den Inhalt der Adresse
?TERMINAL	(- f)	„wahr“, wenn eine Taste gedrückt wurde
COUNT	(addr - addr+1/u)	Wandelt String mit Längenbyte in TYPE-Form um
CR	(-)	Gibt Carriage-Return (CHR$(13)) aus
D.	(d -)	Druckt 32-bit-Zahl
D.R	(d/n -)	Druckt 32-bit-Zahl mit n Stellen
EMIT	(c -)	Gibt ein Zeichen aus
EXPECT	(addr/n -)	Erwartet n Zeichen und legt diese ab addr ab
KEY	(- c)	Liest einen Tastendruck
SPACE	(-)	Gibt ein Space aus
SPACES	(n -)	Gibt n Spaces aus
TYPE	(addr/u -)	Gibt u Bytes ab Adresse aus
WORD	(c -)	Liest im Eingabe-Puffer bis zum nächsten Character c

Zahlenformatierung

#	(d - d)	Wandelt eine Ziffer in String um
# >	(d - addr/u)	Beendet Umwandlung, String kann mit TYPE ausgegeben werden
#S	(d - 0 0)	Wandelt restliche Ziffern in String um
< #	(-)	Eröffnet Zahlenumwandlung in String
HOLD	(c -)	Plaziert Zeichen c an nächster Stelle im Zahlenstring
NUMBER	(addr - d)	Wandelt String ab addr in 32-bit-Zahl um
SIGN	n/d - d)	Fügt Vorzeichen in Ziffernstring ein

Massenspeicher-Befehle

-->	(-)	Compiliert nächsten Screen
B/BUF	(- n)	Gibt Anzahl der Bytes pro Buffer
BLK	(- addr)	Systemvariable (aktuelle Blocknummer)
BLOCK	(n - addr)	Liest Disk-Block nach Adresse
EMPTY-BUFFERS	(-)	Alle Buffer werden als „leer“ gekennzeichnet
FLUSH	(-)	Alle „geänderten“ Blocks werden auf Disk zurückgeschrieben
LIST	(n -)	Listet Screen mit der Nummer n
LOAD	(n -)	Compiliert Screen mit Nummer n
SCR	(- addr)	Systemvariable (aktuelle Screen-Nummer)
UPDATE	(-)	Der zuletzt benutzte Block wird als „geändert“ markiert

Definitionsworte		
: xyz	(-)	Wort xyz wird neu definiert
;	(-)	Ende einer Wortdefinition
<BUILDS DOES>	(-)	Zur Definition von Definitionswörtern
CODE xyz	(-)	Ein Primitive (Maschinenprogramm) wird erzeugt
CONSTANT xyz	(n -)	Eine Konstante xyz mit Wert n wird deklariert
IMMEDIATE	(-)	Das zuletzt definierte Wort wird als immediate markiert
SMUDGE	(-)	Das zuletzt definierte Wort wird kenntlich gemacht
VARIABLE xyz	(n -)	Eine Variable xyz wird deklariert und mit n belegt
Vokabulare		
ASSEMBLER	(-)	Ruft Assembler-Vokabular auf
CONTEXT	(- addr)	Gibt Adresse des Context-Vokabulars (Such-Vokabular)
CURRENT	(- addr)	Gibt Adresse des Current-Vokabulars (Definitions-Voc)
DEFINITIONS	(-)	Setzt CURRENT auf CONTEXT
EDITOR	(-)	Ruft Editor-Vokabular auf
FORTH	(-)	Setzt CONTEXT auf FORTH
VLIST	(-)	Listet alle Wörter im Context-Vokabular
VOCABULARY xyz	(-)	Definiert neues Vokabular xyz
System-Worte		
' xxx	(- addr)	Sucht Adresse des Wortes xxx im Dictionary
(	(-)	Beginn eines Kommentars, Ende mit)
,	(n -)	Legt 16-bit-Zahl im Dictionary ab
ABORT	(-)	Erzwingt Fehler-Abbruch
ALLOT	(n -)	Belegt n Bytes im Dictionary
C,	(b -)	Legt 8-bit-Zahl im Dictionary ab
FORGET abc	(-)	„Vergißt" alle Wörter ab abc (einschließlich abc)
HERE	(- addr)	Zeigt auf nächste freie Stelle im Dictionary
PAD	(- addr)	Puffer 64 Bytes oberhalb von HERE

6/3.4

Programmstrukturen

Nachdem wir uns nun einen Überblick über die wichtigsten Worte von FORTH verschafft haben, ist es an der Zeit, einige Worte für die strukturierte Programmierung zu behandeln. Das wohl grundlegende Wort für Programmstrukturen ist eine „Wenn . . . dann"-Abfrage, die natürlich auch im Wortschatz von FORTH zu finden ist.

Hierzu gleich eine grundsätzliche Eigenschaft von Abfragen in FORTH: Als Anzeige, ob eine Bedingung „wahr" oder „falsch" ist, dient in FORTH wiederum das oberste Stackelement TOS. Ein Wert ungleich Null entspricht dem Zustand „wahr". Die Tests auf Bedingungen hinterlassen im Normall entweder 0, oder -1 (16 binäre Einser) auf dem Stack. Lediglich der Test auf Ungleichheit von zwei Zahlen hinterläßt die Differenz dieser Zahlen.

Doch genug der Theorie, gleich mal ein praktisches Beispiel: Wir wollen ein Wort definieren, das testet, ob eine Zahl durch eine andere teilbar ist und das Ergebnis im Klartext ausgibt. Die Definition ist denkbar einfach:

```
: TEILBAR? MOD IF ." nicht ENDIF ." TEILBAR" ;
```

Was ist hier geschehen? Sehen wir doch mal in unserer Befehlsliste nach: MOD berechnet bei einer Division den Rest, d.h. wenn die Division aufgeht, so ist der Rest Null. Ein Rest erzeugt eine Zahl ungleich Null auf dem Stack. IF sieht sich nun das oberste Stack-Element an (und entfernt es dabei). Ein Wert ungleich Null führt dazu, daß der Programmtext bis zum nächsten ENDIF ausgeführt wird. In unserem Beispiel wird bei Auftreten eines Rests zuerst das Wort „nicht" gedruckt, und dann die Meldung „teilbar". Selbstverständlich kennt FORTH auch noch eine andere (beim BASIC des C 64 leider nicht bekannte Form) der IF-Abfrage, nämlich die IF-Abfrage mit ELSE-Zweig, im Klartext etwa als „wenn . .. dann . . . sonst" formulierbar. Unser Beispiel könnte man also auch so formulieren:

```
: TEILBAR? MOD IF ." NICHT TEILBAR" ELSE ." TEILBAR" ENDIF;
```

Ist die Bedingung „wahr" wird der Teil zwischen IF und ELSE, andernfalls der Teil zwischen ELSE und ENDIF ausgeführt. Manche FORTH-Versionen verwenden anstelle des Wortes ENDIF auch das Wort THEN. Derartige Abfragen lassen sich natürlich auch schachteln.

Ein weiteres Mittel zur Programmstukturierung sind Schleifen. FORTH kennt hier jede denkbare Möglichkeit für Schleifen, fangen wir bei unserer Vorstellung jedoch mit der einfachsten (und in speziellen Fällen eventuell auch sinnvollsten) Schleife an, der Endlosschleife. Das folgende Wort ist ein Beispiel hierfür:

```
: ENDLOS BEGIN ." ICH BIN EINE ENDLOSSCHLEIFE" CR AGAIN ;
```

Es wird beim Aufrufen ständig den Text „ICH BIN EINE ENDLOSSCHLEIFE" ausgeben, solange, bis der Strom ausfällt oder der Programmierer einen Reset herbeiführt. Die Struktur ist offensichtlich, alles zwischen BEGIN und AGAIN wird ewig wiederholt. An dieser Stelle sei noch zu erwähnen, daß Strukturwörter nur in Definitionen erlaubt sind, da sie Sprungbefehle erzeugen. Im normalen Eingabemode würde die schlichte Eingabe von BEGIN ." IRGENDWAS" AGAIN zu einer Fehlermeldung führen.

Als nächstes sind Schleifen zu erwähnen, die durch Auftreten von Bedingungen verlassen werden können. Auch hier ein Beispiel:

```
: BEISPIEL1 BEGIN ." TASTE DRUECKEN" ?TERMINAL UNTIL ;
```

Dieses Wort gibt so lange den Text „TASTE DRUECKEN" aus, bis der Benutzer dieser Aufforderung Folge leistet. Das Wort ?TERMINAL prüft ab, ob eine Taste gedrückt wurde (bei vielen C-64-FORTH-Versionen prüft es den Druck auf die STOP-Taste), ist dies der Fall, dann wird „wahr" auf dem Stack abgelegt. Die Befehle zwischen BEGIN und UNTIL werden also solange wiederholt, bis sie durch UNTIL abgeprüfte Bedingung „wahr" ist, mindestens werden die Befehle jedoch einmal abgearbeitet.

Die andere Abbruchabfrage ist aus folgendem Beispiel ersichtlich:

```
: BEISPIEL2 BEGIN ?TERMINAL NOT WHILE ." TASTE DRUECKEN" REPEAT ;
```

Auch hier wartet das Programm auf einen Tastendruck, die Meldung „TASTE DRUECKEN" wird jedoch nur ausgegeben, wenn ein solcher nicht bereits registriert war. Die Befehle zwischen BEGIN und WHILE werden abgearbeitet und erzeugen die Bedingungsanzeige (Flag) auf dem Stack. Die Befehle zwischen WHILE und REPEAT werden nur abgearbeitet, wenn die Bedingung „wahr" ist, nach REPEAT wird dann zu BEGIN zurückgekehrt und die Bedingung neu abgeprüft. Diese Schleife prüft also die Abbruchbedingung vor der Ausführung und muß daher nicht zwingenderweise mindestens einmal ausgeführt werden.

Schließlich sind noch die Schleifen zu erwähnen, deren Durchlaufzahl bereits feststeht. Eine solche Konstruktion finden wir in DO . . . LOOP. DO erwartet zwei Zahlen auf dem Stack, zuoberst den Startwert der Schleife, als SEC den Endwert der Schleife. LOOP inkrementiert den Index um eins und wiederholt die Schleife,

wenn der Endwert noch nicht erreicht wurde. In folgendem Beispiel würden also die Zahlen 1 bis 10 ausgegeben werden:

```
: ZAHLEN 11 1 DO I . LOOP ;
```

Die Schleife hört auf, wenn der Zähler den Wert 11 erreicht hat und beginnt bei 1. Der Befehl I holt lediglich den aktuellen Schleifenindex auf den Stack, um ihn auszugeben. DO . . LOOP kann auch als DO . . +LOOP verwendet werden, in diesem Fall ist der Inkrement (um wieviel der Index erhöht wird) frei wählbar. Ein Countdown von 10 bis 1 wäre so zu formulieren:

```
: COUNTDOWN 0 10 I. —1 +LOOP ;
```

Hier wird also jedesmal die —1 zum Schleifenindex addiert, solange, bis dieser den Wert 0 erreicht hat. DO . . LOOP-Konstruktionen sind die langsamsten Schleifen in FORTH, aber immer noch um etwa den Faktor 10 schneller als das vergleichbare FOR . . NEXT in BASIC.

Somit sind die wichtigen Strukturwörter von FORTH erklärt. Möchtegern-Strukturwörter wie GOTO sind in FORTH nicht zu finden, wer mit GOTO programmieren will, der hat den Sinn von FORTH noch nicht verstanden. Und GOSUB kann man sich sparen, da jedes FORTH-Wort sowieso ein Unterprogramm darstellt und einfach durch das Nennen seines Namens aufgerufen werden kann.

6/3.5

Arbeitsweise eines FORTH-Compilers

In diesem Abschnitt wollen wir kurz einmal die Arbeitsweise eines FORTH-Systems beleuchten. Bisher konnten wir feststellen, daß FORTH-Programme sehr schnell sind und zudem relativ wenig Speicherplatz benötigen. Nun ist es natürlich interessant zu erfahren, warum dies so ist.

6/3.5.1

Der gefädelte Code

Um zu verstehen, was gefädelter Code ist, wollen wir uns kurz einmal ansehen, wie FORTH seine Programme im Speicher ablegt. Nehmen wir beispielsweise folgende Wortdefinition:

```
: BEISPIEL 10 0 DO I . LOOP ;
```

Wie wir bereits wissen, haben wir mit der „:"-Definition einen neuen FORTH-Befehl definiert, der in diesem Fall „BEISPIEL" heißt.

BEISPIEL gibt die Zahlen von 0 bis 9 am Bildschirm aus. Um uns jetzt den Begriff des gefädelten Codes etwas einfacher vorstellen zu können, stellen wir folgenden Vergleich an:

Jedes FORTH-Wort sei eine Perle. FORTH-Programme, also neue FORTH-Wörter, werden gebildet, in dem man bereits bestehende Wörter (unsere FORTH-„Perlen") nimmt und sie wie auf einen Faden schiebt. In unserem Beispiel haben wir also ein paar FORTH-Befehle zu einem neuen Befehl von größerer Leistung zusammengefaßt. Da FORTH-Wörter nur einen Eingang und einen Ausgang haben, ist der Vergleich mit einer Perle, die ja auch nur einen Eingang und einen Ausgang für den Faden hat, nicht so schlecht.

Selbstverständlich kann unser definiertes Wort in weiteren Definitionen als einfacher Befehl benutzt werden, also auch wieder auf einen neuen imaginären „Programmfaden“ gefädelt werden.

Diese Technik ist nichts neues, BASIC-Programmierer kennen sie unter den Begriffen GOSUB und RETURN. Das RETURN als solches braucht FORTH nicht, es wird automatisch beim Abschluß der Wortdefinition generiert, der ,,;" hat also die Funktion des RETURN-Befehls. GOSUB ist auch unbekannt, um ein Unterprogramm in FORTH aufzurufen, tut man ja nichts anderes, als es bei seinem Namen zu nennen. Jeder BASIC-Programmierer wird hier nun zugeben müssen, daß unser Beispiel aussagekräftiger aussieht als beispielsweise folgendes BASIC-Programm:

```
1000 GOSUB 2100
1010 GOSUB 2200
1020 GOSUB 3500 und so weiter.
```

Da in FORTH das ,,Denken in Unterprogrammen und Programmbausteinen" gefördert wird, ist es nicht weiter verwunderlich, daß FORTH-Programme schnell entwickelt und leicht gewartet werden können. Wer schon einmal versucht hat, in ein 2000 Zeilen langes Spaghetti-Code-BASIC-Programm eine Änderung einzubauen, der weiß was gemeint ist. In FORTH muß dagegen oft nur eine kurze und überschaubare Wortdefinition manipuliert werden.

Nun wollen wir uns jedoch einmal ansehen, wie FORTH-Programme im Speicher abgelegt werden.

6/3.5.2

Abbildung des gefädelten Codes im Speicher

Stellen wir uns einmal vor, FORTH wäre so ,,intelligent" wie unser BASIC-Interpreter (die Anführungszeichen sind voll beabsichtigt), dann würde unser FORTH-Wort BEISPIEL im Speicher etwa so aussehen:

Adresse	Inhalt (Textdarstellung)
$2000	:
$2002	BEISPIEL
$200B	10 0 DO I . LOOP
$201C	;

Nun, so geht's sicherlich nicht. Programme, die wie der Quelltext im Speicher stehen, würden diesen mit nur wenigen Definitionen ausfüllen. Wir wissen aber, daß BASIC für jeden Befehl einen TOKEN (=Schlüsselcode) reserviert. Bei BASIC und seiner beschränkten Befehlszahl genügt hier eine 8-Bit-Zahl (also Werte von 0..255, wobei nur ein kleiner Bereich wirklich für TOKEN benutzt wird). Führen wir für FORTH also zum Spaß mal TOKEN ein, diesmal mit 16-bit Länge. Unser Speicher könnte dann etwa so aussehen:

Addr.	Inhalt	Kommentar
$2000	:	: als Text
$2001	BEISPIEL	Name der Definition
$2009	$0815	TOKEN der Definition
$200B	$0737	TOKEN für 10
$200D	$0685	TOKEN für 0
$200F	$0FCB	TOKEN für DO
$2011	$1B7E	TOKEN für I
$2013	$1007	TOKEN für .
$2015	$1234	TOKEN für LOOP
$2017	$0FFF	TOKEN für ;

Daß dies auch nicht das Gelbe vom Ei ist, sieht wohl jeder. Die TOKEN-Verschlüsselung bedingt nämlich, daß jeder TOKEN erst einmal in einen Befehl umgesetzt werden muß, d.h. man muß die zum TOKEN gehörende Definition erst einmal suchen. Suchen kostet jedoch Zeit, aus diesem einfachen Grund ist BASIC ja so „schnell", jede lächerliche Variable muß gesucht werden, jedes kurze Unterprogramm, einfach fast alles. Trotzdem ist dieses Abbild vom Speicher gar nicht so schlecht, es entspricht fast der Realität. Hier also nun ein „echter" Speicherabzug von einem FORTH-Wort (in Anführungszeichen deshalb, weil die Adreßangaben rein willkürlich gewählt wurden).

Symb.	Addr.	Inhalt	Bemerkung	Pkt.
NFA	$2000	%1PS00100	Statusbyte	(1)
	$2001	BEISPIEL	Name	(2)
LFA	$2009	$1FC2	Verkettung	(3)
CFA	$200B	$0D33	DOCOL-Vektor	(4)
PFA	$200D	$0432	CFA v. CLIT	(5)
	$200F	$0A	Zahl 10	
	$2010	$0737	CFA v. 0	(6)
	$2012	$0822	CFA v. (DO)	(7)
	$2014	$0323	CFA v. I	
	$2016	$0432	CFA v. .	
	$2018	$0689	CFA v. (LOOP)	
	$201A	$FFFA	Offset um -6	(8)
	$201C	$0F3E	NEXT	(9)

In der Spalte „Symb." steht der symbolische Name der Adresse in der Wortdefinition. Doch nun zu den einzelnen „Pkt."-Bemerkungen, die in der Tabelle keinen Platz fanden:

(1) Diese Adresse heißt **N**ame-**F**ield-**A**DRESS, also die Adresse des Namensfeldes. Hier ist ein Statusbyte hinterlegt, das in den unteren 5 Bits die Länge des Wortnamens enthält. Das höchste Bit ist immer auf „1", „S" und „P" sind Statusanzeigen für den FORTH-Compiler.

(2) Hier ist der Name des Wortes als 7-bit-ASCII-Text abgelegt. Manche FORTH-Versionen legen hier nur eine verkürzte Form ab, allerdings ist dann eine Auflistung der Namen mittels **VLIST** nicht mehr möglich. Im letzten Byte des Namens ist das Bit-7 gesetzt.

(3) Diese Adresse heißt **L**ink-**F**ield-**A**ddress, sie dient der Verkettung der einzelnen Wörter im Wörterbuch untereinander. Die zwei Bytes ab dieser Adresse enthalten die NFA des vorhergehenden Wortes im Wörterbuch.

(4) Diese Adresse heißt **C**ode-**F**ield-**A**ddress, hier ist die Adresse der Maschinenbefehle angegeben, die bei Ausführung des Wortes zuerst ausgeführt werden sollen. Hierin können die einzelnen Definitionen unterschieden werden, Colon-Definitionen (:-Definitionen) enthalten hier alle den gleichen Wert. Variablen enthalten einen anderen Wert. Maschinenprogramme (sogenannte Primitives) enthalten hier gleich die Startadresse des Maschinenprogramms.

(5) Diese Adresse heißt **P**arameter-**F**ield-**A**ddress, hier beginnen die Parameter der Wortdefinition. Bei Colon-Definitionen stehen hier jeweils die CFAs der aufgeführten Befehle. Hier ist auch gleich die erste Besonderheit zu entdecken, die Zahl 10 wird als „CLIT 10" abgebildet, da „10" kein eigenes FORTH-Wort ist. Zahlen werden also nicht wie in BASIC als Text abgespeichert, sondern bereits beim Compilieren in die Binärdarstellung umgewandelt.

(6) „0" ist ein definiertes FORTH-Wort, hier ist also nur die CFA von „0" angegeben.

(7) Hier kommt die nächste Besonderheit. Schleifen wie DO..LOOP werden in RUN-Time-Konstruktionen wie (DO) und (LOOP) umgesetzt. Ansonsten gibt es hier nicht viel zu sagen.

(8) Hier ist der Offset hinterlegt, um den zurückgesprungen werden muß, wenn die DO..LOOP-Schleife noch nicht verlassen werden soll.

(9) Hier wird das Wort beendet. Diese Routine entspricht also etwa dem RETURN in BASIC.

Wir sehen, FORTH-Definitionen werden also mit relativ wenig Speicheraufwand abgelegt. Jeder Aufruf eines anderen FORTH-Wortes innerhalb der Definition benötigt im Normalfall zwei Bytes, die die CFA des aufgerufenen Wortes enthalten. Dies ist zum einen optimal speichersparend und zum anderen sehr schnell, da das FORTH-Laufzeitsystem immer sofort weiß, wo es weitergeht. Sehen wir uns also an, wie das Wort abgearbeitet wird.

6/3.5.3

Beispiel für das Abarbeiten einer Wortdefinition

Durch Eingabe des Wortes BEISPIEL wird der Kommando-Interpreter veranlaßt, im Dictionary (Wörterbuch) nach dem Wort zu suchen. Dabei muß er nicht alle Bytes im Dictionary ansehen, sondern kann sich anhand der LFAs von NFA zu NFA durchtasten. Ist das Wort gefunden, so wird die Routine gerufen, deren Adresse in der CFA des Wortes hinterlegt ist. Bei Colon-Definitionen geschieht nun nichts anderes, als daß nun die Wörter, deren CFAs ab PFA hinterlegt sind, abgearbeitet werden. Zum Abschluß wird die Routine NEXT angesprungen, der Kommandointerpreter kehrt in die aufrufende Routine zurück.

Wir sehen also, daß FORTH optimal schnell und speicherplatz-sparend arbeitet. Zuletzt sei noch zu erwähnen, daß es auch sogenannte **Target-Compiler** gibt, die FORTH direkt in die Maschinensprache des Rechners übersetzen. Normale Implementationen arbeiten jedoch mit dem hier beschriebenen Zwischencode, der zum UCSD-p-Code verwandt ist. Dieser Zwischencode läßt sich übrigens mit relativ wenig Aufwand in den Quelltext zurückverwandeln. Man kann also quasi einen Rückübersetzer schreiben, mit dem man sich ansehen kann, wie bereits definierte Wörter aufgebaut sind.

6/4
COMAL

Inhaltsübersicht

6/4.1

COMAL 0.14

Comal ist eine Programmsprache, in der die positiven Aspekte von Basic, Pascal und Logo zusammengefaßt wurden. Die Sprache ist 1973 entstanden, wurde aber erst in der letzten Zeit für Heimcomputer interessant. Wie bei nahezu jedem Namen einer Programmiersprache, versteckt sich auch hinter „Comal“ eine Abkürzung, nämlich „COmon Algorithmic Language“. Die jetzige Version für den C 64 besteht aus einem Programm, das in den Speicher eingeladen wird und dann ein Betriebssystem bildet. Es handelt sich also nicht um einen Compiler, der das geschriebene Programm in Maschinensprache umwandelt. Daraus ergibt sich aber auch ein Nachteil dieser Sprachversion: es sind nämlich nach dem Start des Comal-Programms nur noch 9902 Bytes für den Programmierer frei.

Die eben beschriebene Version ist übrigens ein freies Programm, das heißt, daß es weitergegeben werden darf und daß Sie es sich zum Selbstkostenpreis erwerben dürfen. Dieser preiswerten Version steht eine ROM-Version gegenüber, die dann mehr freien Speicherplatz bieten kann. Da diese aber aus einem Steckmodul besteht, ist sie nicht ganz billig.

Hier soll aber die normale Version, das Comal 0.14, beschrieben werden.

6/4.1.1

Elemente in Comal aus anderen Sprachen

Comal kann nicht als eine völlig neue Programmiersprache bezeichnet werden, weil bei genauerem Hinsehen nur wenige Teile wirklich neu sind. Es wurde vielmehr versucht, alle positiven Seiten einiger Programmiersprachen zusammenzunehmen und daraus eine neue Sprache zu schaffen.

Basic

Basic ist die Grundsprache von Comal, es besteht eine Art Aufwärtskompatibilität von Basic zu Comal, das heißt, daß jeder, der Basic einigermaßen beherrscht, auch mit Comal umgehen kann. Nach und nach wird er sich dann die Feinheiten von Comal aneignen. Es wird oft gesagt, daß Basic einen Programmierer verderbe, diese Elemente wurden aber in Comal trotz der nahen Verwandtschaft ausgeglichen. Dem etwas versierten Basic-Programmierer wird auffallen, daß es in Comal keine Möglichkeiten gibt, einen Befehl abzukürzen. Das kommt, weil es die Möglichkeit gibt, neue Befehle zu definieren.

Pascal

Von Pascal hat Comal besonders die Möglichkeit der strukturierten Programmierung übernommen, die noch in einem eigenen Kapitel beschrieben wird. Strukturierte Sequenzen sind leichter zu programmieren als die in Basic unumgängliche Umschreibung mit umständlichen „if"-Befehlssequenzen. Mit Befehlen wie zum Beispiel „repeat . . . until" oder „case . . . when . . :" werden Befehlswörter gebraucht, die der menschlichen Sprache und Denkweise näher sind als eine Umschreibung durch „if". Das Programm wird dadurch leichter lesbar und es kann nicht passieren, daß man, wie in Basic, sein eigenes Programm nicht mehr lesen kann. Es ist ja schließlich neben der Lösung einer Problemstellung auch das Ziel eines Programmierers sein Programm übersichtlich und gut lesbar zu gestalten; das ist durch diese Elemente aus Pascal gut möglich.

Logo

Wesentliche Anteile hat auch die Programmiersprache Logo an Comal. Es besteht nämlich auch bei Comal die Möglichkeit, neue Befehle zu definieren, beziehungsweise Unterprogramme mit Namen aufzurufen. Daneben können auch Funktionen definiert werden; und dies nicht nur, wie in Basic, in einer Zeile, sondern als Unterprogramm in beliebiger Länge. Auch dies steigert die Übersichtlichkeit und vor allem die Lesbarkeit eines Programmes enorm. Mußte man in Basic die Funktion eines Unterprogrammes mit Hilfe einer „rem"-Zeile deutlich machen, so kann man in Comal das Unterprogramm mit einem Namen aufrufen, der seiner Funktion entspricht.

Es wurde aber auch die bekannte Turtle-Grafik von Logo übernommen. Sie ist recht einfach zu programmieren und ist der menschlichen Denkweise angepaßt. Im einfachsten Fall gibt es hier einen Zeichenstift (Turtle), der um seine eigene Achse gedreht und vorwärts und rückwärts bewegt werden kann. Der Name Turtle wurde von dem zum Zeichnen anfangs verwendeten schildkrötenähnlichen Gebilde übernommen. Eine detailliertere Beschreibung dieser Grafik folgt in einem eigenen Kapitel.

6/4.1.2

Fehlerbehandlung

Wie schon erwähnt, ist der Speicherplatz des C 64 mit Comal sehr knapp. Aus diesem Grund wurden die Texte der Fehlermeldungen auf einer Datei auf der Diskette abgelegt und jedesmal, wenn ein Fehler auftritt, wird dieser Text aus der Datei gelesen. Bei Comal wird jede Zeile sofort beim Eingeben auf ihre syntaktische Richtigkeit hin geprüft. Verschreibt man sich und gibt die falsche Anweisung ein, dann läuft das Laufwerk an; dies kann einem manchmal ziemlich auf die Nerven gehen. Deshalb gibt es den Befehl „SETMSG-", der bei einem Fehler nur noch die Nummer des Fehler ausgibt. Will man also flüssig mit Comal arbeiten, dann kann man sich eine Liste der Fehler und ihrer Nummern ausdrucken lassen und jeden auftretenden Fehler auf dieser Liste nachschauen. Will man wieder auf die normale Fehlerausgabe umschalten, so kann man dies mit dem Befehl „SETMSG+" tun.

6/4.1.3

Formate laden und speichern

Comal-Programme können in zwei verschiedenen Formaten abgespeichert werden. Zuerst gibt es das normale Format, bei dem die Befehle, wie in Basic, in verkürzter Form kodiert abgespeichert werden. Diese Kodierung ist speziell für das Comal 0.14-Format. Dazu stehen die aus Basic bekannten Befehle „LOAD" und „SAVE" zur Verfügung, wobei die Geräteadresse des Laufwerks nicht angegeben werden muß. Man muß also nicht, zum Beispiel, LOAD „grafik",8, eingeben, sondern man braucht nur LOAD „grafik" einzugeben. Zusätzlich ist noch der Befehl „CHAIN" vorhanden, mit dem Programme geladen und danach automatisch gestartet werden können. Seine Syntax ist mit der des Befehls „LOAD" identisch.

In Basic ist es nur sehr schwierig möglich, Programme verschiedener Basic-Dialekte untereinander auszutauschen. Um dem in Comal entgegenzuwirken, können Pro-

gramme in Textform abgespeichert werden. Die Befehle werden nicht kodiert, sondern als Text auf dem Datenträger abgelegt werden. Dadurch kann diese Textdatei auch von einer anderen Comalversion geladen und danach wieder im Speicher kodiert werden. Der Befehl zum Abspeichern lautet hier „LIST <Programmname>“ und der Befehl zum Einlesen „ENTER <Programmname>“. Programme, die in diesem Format abgelegt durch „ENTER“ eingelesen werden, werden in ein eventuell schon im Speicher befindliches Programm eingefügt. Der Befehl „enter“ wirkt also ähnlich wie der in Basic verbreitete Befehl „MERGE“. Diese Textdateien sind durch die fehlende Kodierung erheblich länger als die kodierten Programmdateien.

6/4.1.4

Strukturiertes Programmieren

Strukturiertes Programmieren ist ein übersichtliches, einfaches und immer gut lesbares Programmieren. All das wird in Basic oft vernachlässigt und deshalb sollte man sich als reiner Basic-Programmierer auch mit dem strukturierten Programmieren einmal ernsthaft beschäftigen.

6/4.1.4.1

Strukturiertes Listen

Wie in Basic gibt es auch in Comal den Befehl „LIST“. Mit ihm kann das im Speicher befindliche Programm ausgegeben werden. Folgende Formate sind wie in Basic möglich:

list	Zeigt das ganze Programm
list zn1-	Zeigt das Programm ab Zeilennummer zn1
list -zn1	Zeigt das Programm bis zn1
list zn1-zn2	Zeigt das Programm von zn1 bis zn2

Während des Listens kann die Ausgabe, ebenso wie in Basic, durch die „CONTROL"-Taste verzögert werden. Zusätzlich kann der Ausgabevorgang mit einem Betätigen der Leertaste angehalten und mit einem weiteren Druck auf diese Taste fortgesetzt werden.

Der größte Unterschied zu dem Basic-Listbefehl ist allerdings, daß das Programm strukturiert ausgegeben wird. Das bedeutet, daß überall, wo Strukturbefehle wie Schleifen oder Bedingungsbefehle verwendet werden, der Teil des Programms, der innerhalb einer solchen Struktur steht, um ein Zeichen nach rechts verschoben wird. Das sieht dann zum Beispiel so aus:

Basic	Comal
1 FOR a=1 TO 100 2 PRINT „Das ist der"; a;". Durchlauf der Schleife." 3 NEXT a	1 FOR a:=1 to 100 DO 2 PRINT „Das ist der"; a;". Durchlauf der Schleife." 3 ENDFOR a

Dadurch wird das Programmlisting natürlich viel übersichtlicher und einfacher zu lesen. Auch ineinander verschachtelte Strukturen werden sofort deutlich, weil sie dann jeweils um mehrere Zeichen nach rechts eingeschoben werden. Diesem strukturierten Listen kommt besonders deshalb eine große Bedeutung zu, weil es in Comal erheblich mehr Strukturbefehle gibt als in Basic.

Es ist aber auch möglich, das Programm nicht strukturiert, sondern wie in Basic auszugeben. Das wird mit dem Befehl „EDIT" erreicht, der die gleiche Syntax wie der Befehl „LIST" hat, der aber das Programm so ausgibt, daß Strukturen nicht mehr äußerlich am Listing zu erkennen sind.

6/4.1.4.2

Programmschleifen

Programmschleifen sind elementare Mittel des Programmierens, die jeder Rechner beherrscht. Allerdings gibt es hier verschiedene Ausprägungen dieser Fähigkeit. In Basic zum Beispiel, gibt es nur die FOR . . . NEXT-Schleife. Das ist sehr wenig, und dem wurde in Comal abgeholfen. In dieser Programmiersprache kann man nämlich lästige Umschreibungen vergessen.

FOR . . . STEP . . . ENDFOR

Wie in Basic, ist es auch in Comal möglich, Schleifen zu programmieren. Im Unterschied zu Basic heißt hier der Schleifenbegrenzungsbefehl nicht „NEXT“, sondern „ENDFOR“. Wird trotzdem „NEXT“ verwendet, setzt Comal automatisch dafür „ENDFOR“ ein. Beim nächsten Listen erscheint also an dieser Stelle „ENDFOR“, um die Aufwärtskompatibilität von Basic zu Comal zu wahren. Die „FOR“-Anweisung muß in einer eigenen Zeile stehen; am Ende dieser Zeile steht „DO“, um die Anweisungsliste einzuführen. Dieses „DO“ muß aber nicht eingegeben werden, sondern es wird auch automatisch von Comal eingefügt. Die Schleifenstruktur sieht also folgendermaßen aus:

```
FOR <Schleifenvariable>:=<Anfangswert> TO <Endwert> (STEP <Schrittweite>) DO
...
<Schleifenanweisungen>
...
ENDFOR (<Schleifenvariable>)
```

REPEAT . . . UNTIL

Mit der REPEAT . . . UNTIL-Anweisung kann eine Schleife beschrieben werden, die beim Programmieren relativ oft Verwendung findet, aber in Basic nur durch eine Umschreibung realisiert werden kann. Diese Anweisung sieht im Programm folgendermaßen aus:

```
REPEAT
...
<Schleifenanweisungen>
...
UNTIL <Bedingung>
```

Die Anweisungen innerhalb einer Schleife werden so oft wiederholt, bis eine bestimmte Bedingung wahr wird, die nach „UNTIL“ steht. Hierzu ein Beispiel:

```
REPEAT
INPUT „Bitte Codewort eingeben: „:code$
UNTIL code$=„comal"
```

Der Computer bittet Sie hier solange, das Codewort einzugeben, bis Sie das richtige Codewort, nämlich „comal“, eingegeben haben.

WHILE . . . ENDWHILE

Auch die while . . . endwhile-Anweisung dient dazu, Programmideen, die nahezu aus der Alltagssprache kommen, möglichst einfach auf dem Computer realisieren zu können. Im Programm wird die Anweisung so verwendet:

```
WHILE <Bedingung>
. . .
<Schleifenanweisungen>
. . .
ENDWHILE
```

Die Schleifenanweisungen werden solange ausgeführt, wie die nach „WHILE“ aufgeführte Bedingung wahr ist. Ist diese Bedingung nicht mehr wahr, dann wird das Programm in der Zeile nach dem Begrenzungsbefehl „ENDWHILE“ fortgesetzt.

Es ist aber auch eine einzeilige Kurzform dieser Anweisung verfügbar, die keinen Begrenzungsbefehl benötigt. Sie wird so verwendet:

```
WHILE <Bedingung> DO <Schleifenanweisung>
```

Wenn es sich nur um eine Schleifenanweisung handelt, kann diese Kurzform benutzt werden. Nachdem die Bedingung in der Anweisung unwahr geworden ist, wird das Programm in der nächsten Zeile fortgesetzt.

6/4.1.4.3

Verzweigungen

Auch in COMAL sind Verzweigungen beziehungsweise Bedingungsbefehle verfügbar. Im Gegensatz zu Basic können solche Bedingungsstrukturen mehrzeilig sein.

Der Trend geht also hier weg von einer Verzweigung in einen eigenen Programmteil (mit „GOTO"); es wird versucht, bestimmte Strukturen, die bei einer wahren Bedingung ausgeführt werden sollen, gleich an Ort und Stelle zu bearbeiten. Dadurch wird ein in Basic zwangsweise vorhandenes Herumspringen weitgehend vermieden.

IF ... ELSE ... ELIF ... ENDIF

Wie auch in Basic ist der Bedingungsbefehl „IF" vorhanden. In Basic ist dieser nur einzeilig verfügbar; will man aber trotzdem mehrzeilige Befehlsfolgen unterbringen, so muß dies umständlich umschrieben werden. Das ist in COMAL nicht nötig, aber die magere einzeilige Variante des Befehls ist wegen der Aufwärtskompatibilität trotzdem vorhanden. Die Syntax sieht, der von Basic entsprechend, folgendermaßen aus:

```
IF  <Bedingung>  THEN  <Anweisung>
```

Die einfachste Form der Syntax in der mehrzeiligen Version lautet folgendermaßen:

```
IF <Bedingung> THEN
   <Anweisung 1>
   <Anweisung 2>
   ...
   <Anweisung n>
ENDIF
```

Man muß also lediglich nach „THEN" eine neue Zeile anfangen und die Anweisungsliste dann mit „ENDIF" abschließen. Die komplexeste Form sieht dann so aus:

```
IF <Bedingung> THEN
   <Anweisungen>
ELIF <Bedingung> THEN
   <Anweisungen>
ELIF <Bedingung> THEN
   <Anweisungen>
ELSE
   <Anweisungen>
ENDIF
```

Wie man sieht, können beliebig viele „ELIF"-Anweisungen eingebaut werden. Die Auswertung dieser Struktur geht dann wie folgt vor sich: Es wird zuerst die „IF"-Bedingung überprüft. Ist diese wahr, dann werden die Anweisungen unter „IF" ausgeführt und das Programm wird nach „ENDIF" fortgesetzt. Trifft die „IF"-Bedingung nicht zu, dann werden nacheinander die „ELIF"-Bedingungen durchgeprüft. Kommt der Computer zu einer, die wahr ist, dann führt er die zugehörigen Anweisungen aus und macht dann im Programmablauf nach „ENDIF" weiter. Ist die „IF"- und sind alle „ELIF"-Bedingungen unwahr, dann werden die Anweisungen nach dem „ELSE"-Befehl ausgeführt. Hierzu ein Beispiel:

```
INPUT „Bitte geben Sie eine Zahl ein :":zahl
IF zahl <100 THEN
  PRINT „Die Zahl ist kleiner als 100"
ELIF zahl<200 THEN
  PRINT „Die Zahl liegt zwischen 100 und 200"
ELIF zahl<300 THEN
  PRINT „Die Zahl liegt zwischen 200 und 300"
ELIF zahl<400 THEN
  PRINT „Die Zahl liegt zwischen 300 und 400"
ELSE
  PRINT „Die Zahl ist größer als 399"
ENDIF
END
```

Zu beachten ist noch einmal, daß, wenn eine „IF"- oder „ELIF"-Anweisung wahr ist, keine andere „ELIF"-Anweisung mehr überprüft wird.

CASE ... WHEN ... OTHERWISE ... ENDCASE

Die „CASE"-Anweisung funktioniert ähnlich wie die „IF"-Anweisung, nur wurde hier zusätzlich versucht, den Wortlaut und die Form der Programmstruktur der menschlichen Sprache anzupassen. Die allgemeine Form der „CASE"-Anweisung:

```
CASE <Ausdruck> OF
WHEN <Werte1>
  <Anweisungen>
WHEN <Werte2>
  <Anweisungen>
OTHERWISE
  <Anweisungen>
ENDCASE
```

Natürlich können auch bei der „CASE"-Anweisung Teile weggelassen werden. Mindestens muß aber der „CASE"- und der „ENDCASE"-Befehl vorhanden sein. Bei der „CASE"-Anweisung wird geprüft, ob der Ausdruck, der aus einem numerischen Wert oder einer Zeichenkette bestehen kann, bestimmte Werte hat. Dieser Ausdruck wird hinter dem „CASE"-Befehl plaziert. Danach folgen hinter „WHEN" ein oder mehrere Werte; bei mehreren Werten, die für die Bedingung zur Auswahl stehen sollen, müssen diese mit Komma getrennt sein. Enthält der Ausdruck einen dieser Werte, dann werden die Anweisungen in den folgenden Zeilen ausgeführt. Das Programm wird dann nach der „ENDCASE"-Anweisung fortgesetzt. Hat nun aber der Ausdruck keinen der hinter den „WHEN"-Anweisungen angebotenen Werte, dann werden die Anweisungen hinter der „OTHERWISE"-Anweisung ausgeführt. Hierzu ein Beispiel:

```
. . .
INPUT „Soll ich diese Anweisung ausführen:“:antw$
CASE antw$ of
WHEN „j“, „ja“, „y“, „yes“
  print „Anweisung wird ausgeführt.“
. . .
WHEN „n“, „nein“, „no“
  print „Anweisung wird nicht ausgeführt.“
  . . .
OTHERWISE
  print „Eingabe nicht verstanden.“
  print „Bitte wiederholen.“
  . . .
ENDCASE
```

Zuerst wird gefragt, ob eine bestimmte Anweisung ausgeführt werden soll. Danach kommt die Auswertung mit „CASE . . . “ . Ist die Antwort „j“, „ja“, „y“ oder „yes“, dann wird die Anweisung ausgeführt. Ist die Antwort „n“, „nein“ oder „no“, dann wird die Anweisung nicht ausgeführt. Hat die Antwort keinen der sieben zur Auswahl gestellten Werte, dann werden die Anweisungen nach „OTHERWISE“ ausgeführt (hier nur angedeutet).

6/4.1.5

Unterprogramme und Variablen

Im Gegensatz zu Basic, können in COMAL neue Befehle und Funktionen in abgeschlossenen Unterprogrammstrukturen definiert werden. Auch die verschiedene Handhabung von Variablen ist für Basic-Benutzer neu. Während es in Basic nur globale Variablen gibt, sind in COMAL globale und lokale Variablen verfügbar.

6/4.1.5.1

Definition neuer Befehle (Prozeduren)

In COMAL ist es im Gegensatz zu Basic möglich, neue Befehle zu definieren. Vereinfacht können diese Befehle dazu verwendet werden, ein Unterprogramm nicht mit einer Zeilennummer, sondern mit einem Label, also einem Namen, anzuspringen. Wenn diese Funktion als Möglichkeit zur Definition neuer Befehle genutzt wird, können, wie bei normalen Befehlen auch, Parameter, also Variablen, an das Unterprogramm übergeben werden. In anderen Programmiersprachen wird diese Struktur auch Prozedur genannt (z.B. bei PL/1 und Pascal). Im Standardfall sieht die Befehlsfolge folgendermaßen aus:

```
PROC <Name der Struktur> (<Liste von Variablen>)
 . . .
  <Anweisungsliste>
 . . .
ENDPROC <Name der Struktur>
```

Zum besseren Verständnis nun ein einfaches Beispiel:

```
INPUT „Wieviele Zeilen Vorschub:":zeilen
ZEILENVORSCHUB (zeilen)
PRINT „Vorschub ausgeführt."
. . .
. . .
PROC ZEILENVORSCHUB (vorschub)
  FOR a:=1 to vorschub do
    PRINT
  ENDFOR a
ENDPROC ZEILENVORSCHUB
```

In diesem Programmbeispiel wird zuerst gefragt, wieviele Zeilen vorgeschoben werden sollen. Der eingegebene Wert wird in der Variablen „vorschub" gespeichert. Danach wird der neue Befehl „ZEILENVORSCHUB" mit der Variablen „vorschub" angesprochen. Dieser ist im darauffolgenden Unterprogramm definiert. Hier wird der übergebene Parameter verarbeitet. Dann werden entsprechend der Variablen „vorschub" Vorschübe durch eine Schleife mit PRINT-Anweisung ausgeführt. Wird der ENDPROC-Befehl erreicht, wird der Programmablauf im Hauptprogramm fortgesetzt.

Wird das Sprachelement „REF" bei der Definition des Befehles vor die Variable im Argument gesetzt (z.B. PROC ZEILENVORSCHUB [REF vorschub]), dann werden innerhalb der Prozedur bei der Ausführung nicht die aktuellen Werte der Parameter, sondern die Namen der Parameter verwendet. Vor allem wird dies wichtig, wenn den

aktuellen Variablennamen beim Aufruf der Prozedur innerhalb der Prozedur Werte zugeordnet werden sollen.

6/4.1.5.2

Definition neuer Funktionen

In nahezu identischer Weise können auch Funktionen wie Befehle definiert werden. Im Argument dürfen dabei beliebig viele Variablen stehen und die Sequenz kann beliebig lang werden. Der in der Prozedur errechnete Funktionswert wird mit dem Befehl „RETURN <Variable>“ an das Hauptprogramm zurückgegeben. Hierzu ein Beispiel:

```
INPUT „Zahl“:z
PRINT z
PRINT bruchteil(z)
...
FUNC bruchteil(a)
  br=a-int(a)
  return(br)
ENDFUNK bruchteil
```

Im ersten Teil dieses Beispiels wird die Eingabe einer Zahl verlangt, die in der Variablen „z“ gespeichert und dann sofort auf dem Bildschirm ausgegeben wird. Danach wird mittels eines PRINT-Befehls die Funktion bruchteil() angesprochen, die im folgenden definiert ist. Zu Beginn steht der Befehl „FUNC“ und danach der Name der Funktion und die Funktionsvariable, wie sie in diesem Teilprogramm benutzt wird. Dann wird die Funktion angegeben und der Wert mit „RETURN“ an das Hauptprogramm zurückgegeben. Am Ende steht „ENDFUNC“ mit dem Funktionsnamen, um die Funktion räumlich abzuschließen.

6/4.1.5.3

Lokale und globale Variablen (CLOSED)

In Basic gibt es nur globale Variablen, das sind Variablen, die jederzeit und von jedem Programmabschnitt aus abrufbar sind. Lokale Variablen kommen bei Befehls- und Funktionsdefinitionen vor. Die Funktionsargumente sind immer lokal:

```
PROC TEILEN (namen$)
FUNC PRIMZAHL (zahl)
```

Dies bedeutet, daß die gekennzeichneten Variablen auch im Hauptprogramm vorkommen können, aber daß sich Ihr Wert auch nicht ändert, wenn die Werte der Funktionsargumente im Unterprogramm geändert werden. Die beiden Variablen sind also trotz des gleichen Namens unabhängig. Hierzu ein Programmbeispiel:

```
a=3;b=4
PRINT „a=";„b=",b
produkt(a)
PRINT „a=",a;„b=",b
. . .
PROC produkt(a)
  a=a*b;b=5
  PRINT „a=",a;„b=",b
ENDPROC produkt
```

Dabei erhält man folgendes Ergebnis:

```
a=3      b=4
a=12     b=5
a=3      b=5
```

Der Wert von „a" wird im Unterprogramm geändert. Trotzdem hat „a" nachher im Hauptprogramm wieder den gleichen Wert wie ganz am Anfang. „a" war also im Unterprogramm eine lokale Variable. Auch der Wert von „b" ist im Unterprogramm abgeändert worden. Allerdings hat „b" nachher im Hauptprogramm noch diesen abgeänderten Wert. „b" ist also eine globale Variable (wie in Basic immer).

Es gibt aber auch die Möglichkeit, alle im Unterprogramm ausgegebene Variablen zu lokalen Variablen zu machen. Dies ist mit Hilfe des Befehls „CLOSED" möglich, der dann am Ende der Einleitungszeile steht, wie zum Beispiel:

```
PROC produkt(a) CLOSED
```

Wenn man das obige Programm so ändern würde, dann wäre auch „b" im Unterprogramm eine lokale Variable. Der Programmablauf sähe nun so aus:

```
a=3     b=4
a=12    b=5
a=3     b=4
```

Nun hat auch „b" trotz einer Änderung seines Wertes im Unterprogramm im Hauptprogramm wieder den gleichen Wert wie ganz am Anfang. „b" ist nun im Unterprogramm eine lokale Variable.

6/4.1.6

Hochauflösende Grafik

Der Commodore 64 ist mit einer für seine Größe recht guten Auflösung versehen. Diese wird aber in Basic durch keinen einzigen Befehl unterstützt. Dem ist in COMAL durch eine ganze Reihe von Grafikbefehlen abgeholfen worden.

Mit dem Befehl SETGRAPHIC können Sie die Grafik einschalten. Geben Sie SETGRAPHIC 0 ein, dann arbeiten Sie auf einem hochauflösenden Bildschirm mit zwei Farben (Vordergrund und Hintergrund) mit einer Punktmatrix von 320x200. Mit SETGRAPHIC 1 arbeiten Sie mit einem Mehrfarbenbildschirm mit vier Farben (3 Vordergrund, 1 Hintergrund) mit einer Auflösung von 160x200 Punkten.

Wenn sie die Grafik schon einmal initiiert hatten, genügt die Eingabe von SETGRAPHIC ohne Argument, um wieder auf den vorher festgelegten Grafikbildschirm zu kommen.

6/4.1.6.1

Aufbau des Bildschirms und Modi

Wie schon erwähnt ist die Auflösung vom Grafikmodus abhängig. Bei manchen Grafikbefehlen ist ein Punkt direkt adressierbar; hierbei ist zu beachten, daß sich in der linken unteren Ecke der Punkt mit den Koordinaten 0,0 (im Gegensatz zur „normalen" Koordinatenverteilung beim C 64) befindet. Die Ausdehnung nach oben geht mit 200 Punkten bis zum Punkt 0,199 in der linken oberen Ecke. Zu bemerken ist, daß bei allen folgenden Punktangaben zuerst die x-Koordinate und danach mit Komma abgetrennt die y-Koordinate angegeben wird. Nach rechts beträgt die Ausdehnung je nach Modus 320 oder 160 Punkte bis zum Punkt in der rechten unteren Ecke mit den Koordinaten 319,0 bzw. 159,0.

Mit dem Befehl SETTEXT kann vom Grafikbildschirm wieder auf den Textbildschirm zurückgeschaltet werden. Auf dem Grafikbildschirm selbst gibt es zwei verschiedene Modi der Aufteilung. Geben Sie den Befehl FULLSCREEN ein, dann ist der Grafikbildschirm über den ganzen verfügbaren Bildschirm verteilt. Besonders im Direktmodus ist es aber interessant, den Befehl SPLITSCREEN einzugeben, wodurch am oberen Bildschirmrand zwei Textzeilen eingeblendet werden, auf denen Direktanweisungen eingegeben werden können. Im Vierfarbenbetrieb ist dies jedoch nicht möglich.

6/4.1.6.2

Idee der Turtle-Grafik

Die Turtle-Grafik, die auch in der Programmiersprache Logo verwendet wird, ist eine sehr einfache und leicht verständliche Handhabung der Grafik. Sie wurde ursprünglich für Kinder entworfen, wobei hier allerdings eine erweiterte Version vorliegt.

Nach dem Einschalten der Grafik erscheint in der Mitte des Bildschirms ein Dreieck, das Turtle oder, zu deutsch, Schildkröte genannt wird. Mit einfachen Befehlen

kann diese Schildkröte nun um eine jeweils bestimmte Schrittanzahl gedreht, vorwärts, rückwärts oder nach links oder rechts bewegt werden. Zu diesem elementaren Befehlen sind noch einige Befehle ergänzt worden, mit denen man die Schildkröte auch direkt zu einem Punkt bewegen kann. Schließlich kann vor jedem Bewegen der Turtle noch festgelegt werden, ob sie nur über den Bildschirm bewegt werden oder ob sie dabei auch eine Linie zeichnen, das heißt quasi eine Spur hinterlassen, soll.

6/4.1.6.3

Die Grafik-Befehle

Im folgenden werden nun alle noch nicht erwähnten Grafikbefehle und solche, die mit der Grafik etwas zu tun haben, in alphabetischer Reihenfolge aufgeführt und kurz erklärt.

BACK <numerischer Ausdruck>

Mit diesem Befehl kann die Schildkröte um die nach BACK stehende Schrittanzahl nach hinten bewegt werden. Es handelt sich also hier um eine relative Anweisung zur aktuellen Position.

CLEAR

Mit dem Befehl CLEAR wird der Grafikbildschirm gelöscht, das heißt, daß alle Punkte in Hintergrundfarbe (BACKGROUND!) dargestellt werden.

DRAWTO <x-Koordinate>, <y-Koordinate>

Mit DRAWTO kann die Schildkröte von einer beliebigen Position aus an die mit Koordinaten angegebene Stelle bewegt werden (absolute Positionierung). Dabei wird hier immer eine Linie gezeichnet.

FILL <x-Koordinate>, <y-Koordinate>

Dieser Befehl füllt, von den angegebenen Koordinaten ausgehend, eine Fläche auf dem Bildschirm aus. Diese kann zum Beispiel mit vorher gezeichneten Linien vorgegeben werden. Sobald die Begrenzung um den angegebenen Punkt irgendwo „undicht“ ist, wird auch außerhalb der gewünschten Fläche ausgefüllt.

FORWARD <Schritte>

Mit dieser Anweisung können Sie die Schildkröte um eine bestimmte Anzahl von Schritten, bzw. Bildschirmpunkten, nach vorne (=vorwärts) bewegen.

HIDETURTLE

Mit HIDETURTLE wird die Schildkröte, die die Position des Grafikkursors angibt, unsichtbar gemacht, um zum Beispiel eine fertige Grafik auch ohne Schildkröte betrachten zu können.

HOME

Diese Anweisung bringt die Schildkröte in die Mitte des Grafikbildschirms. Sollte mit FRAME ein Rahmen definiert sein, so wird sie in die Mitte dieses Rahmens gebracht.

LEFT <Winkelgrade>

Mit diesem Befehl wird die Schildkröte um die angegebene Anzahl von Graden nach links gedreht. Sie würde sich also zum Beispiel beim nächsten FORWARD-Befehl in eine andere Richtung bewegen. Eine ganze Drehung entspricht also 360 Grad.

MOVETO <x-Koordinate>, <y-Koordinate>

Mit MOVETO kann die Schildkröte zu den gewünschten Koordinaten verschoben werden, ohne daß sie eine Linie zeichnet. Es handelt sich hier also um das Gegenstück zu DRAWTO.

PENDOWN

Mittels dieser Anweisung kann die Schildkröte auf Zeichenmodus eingestellt werden. Das bedeutet, daß bei allen Befehlen wie FORWARD oder BACK eine Linie gezeichnet wird. Der imaginäre Schreibstift (pen) wird auf die Schreibfläche herunter (down) gesetzt.

PENUP

Dieser Befehl ist das genaue Gegenstück zu PENDOWN. Wird nach ihm durch einen Befehl wie FORWARD oder BACK die Schildkröte bewegt, dann wird keine Linie gezeichnet, bzw. es wird keine „Spur hinterlassen".

PLOT <x-Koordinate>, <y-Koordinate>

Mit PLOT wird an der angegebenen Stelle auf dem Grafikbildschirm ein Punkt gezeichnet.

PLOTTEXT <x-Koordinate>, <y-Koordinate>, <Text>

Durch diese Anweisung kann auch auf dem Grafikbildschirm normaler Text dargestellt werden. Die angegebenen Koordinaten bestimmen den Punkt, an dem der Text anfängt; der zu druckende Text wird einfach hinter den Koordinaten in Form einer Zeichenkette abgelegt. Die Anfangskoordinaten werden vom Computer so gerundet, daß der Text in den vom normalen Bildschirm her bekannten Zeilen und Spalten gedruckt wird. Diese Anweisung ist im Vierfarbenmodus nicht verwendbar.

RIGHT <Winkelgrade>

Mit diesem Befehl kann die Schildkröte, wie bei LEFT erklärt, um eine angegebene Gradzahl nach rechts gedreht werden.

SETHEADING <Winkelgrade>

Durch SETHEADING kann die Schildkröte auf eine bestimmte Richtung gedreht werden, wobei 0 z.B. 360 Grad nach oben und 90 Grad nach rechts zeigt.

SETXY <x-Koordinate>, <y-Koordinate>

Mit dieser Anweisung ist es möglich, die Schildkröte an einen durch die Koordinaten festgelegten Platz zu bringen. Im Gegensatz zu DRAWTO und MOVETO ist es aber hier entscheidend, ob man sich im PENUP- oder im PENDOWN-Modus befindet.

SHOWTURTLE

Hier handelt es sich um das Gegenstück zum Befehl HIDETURTLE. Nachdem die Schildkröte mit HIDETURTLE unsichtbar gemacht worden ist, kann sie mit SHOWTURTLE wieder sichtbar gemacht werden.

TURTLESIZE <Größe>

Mit dieser Anweisung kann der Schildkröte eine bestimmte Größe gegeben werden. Diese Größe reicht von 0 bis 10, wobei 10 der größte Wert ist und auch der, der beim Einschalten von COMAL eingestellt ist.

Schalten sie also mit SETGRAPHIC 0 im Direktmodus den Grafikmodus an und geben Sie zum Beispiel TURTLESIZE 5 ein und sehen Sie, was passiert. Die Schildkröte hat ihre Größe halbiert. Mit TURTLESIZE 10 bekommt sie wieder die ursprüngliche Größe.

Nun folgen drei Befehle, mit deren Hilfe man die Farben in verschiedenen Bereichen des Bildschirms einstellen kann.

1. BACKGROUND <Farbnummer>

Mit BACKGROUND kann die Hintergrundfarbe, das ist die Farbe, auf der die eigentliche Grafik gezeichnet wird, geändert werden. Die zugehörige Nummer zu einer Farbe finden Sie in Ihrem Bedienungshandbuch.

2. BORDER <Farbnummer>

Mit dieser Anweisung können Sie dem Feld außerhalb der Fläche, in der gezeichnet werden kann, also im Rahmen, eine neue Farbe einsetzen.

3. PENCOLOR <Farbnummer>

Durch diesen Befehl können Sie die Zeichenfarbe ändern. Geben Sie also hiermit eine neue Farbe an und zeichnen Sie danach eine Linie, dann erscheint diese in der neuen Zeichenfarbe.

Teil 7

Hard- und Softwareergänzungen

7/1

Handelsware

7/1.8

Maussteuerungen

Neben der in diesem Buch in Kapitel 4/4.6 vorgestellten Maussimulation mit dem Joystick, möchten wir in Teil 7 auf handelsübliche Mäuse eingehen. Bei allen hier vorgestellten Mäusen ist jedoch eine Mauseigenschaft zu berücksichtigen, die ihre Anwendung im Home-Computerbereich stark beeinträchtigt: Der Platz. Wie alle Tiere braucht auch die Maus ihren Auslauf.

7/1.8.1

Magic-Maus

Eine der beiden bei Redaktionsschluß auf dem Markt befindlichen Mäuse war die Magic-Maus von CONTRIVER ENTERPRISE CO., LTD. In Deutschland wird das Produkt vertrieben von TS ELEKTRONIK, 6657 Gersheim. Im Handel ist sie für ca. DM 210,— zu haben.

Zum Lieferumfang gehört neben der Maus ein 23-seitiges englisches Handbuch und eine sechsseitige deutsche auszugsweise Übersetzung, die aber alles wesentliche enthält. Selbstverständlich ist eine Diskette mit der notwendigen Treibersoftware vorhanden, die zudem noch ein Zeichenprogramm und Hilfsmittel zur Sprite- und Icon-Erstellung beinhaltet.

Die Ladezeiten der einzelnen Teilprogramme liegen im üblichen Rahmen bei Verwendung einer 1541. Nachdem das Hauptprogramm mit

```
LOAD „MENU“, 8
```

geladen wurde, ist dieses mit RUN zu starten. Damit wird ein weiteres Programm geladen, das schließlich in einem Menü am Bildschirm den verwendeten Drucker erfragt. Es stehen vier Druckertypen zur Auswahl:

— EPSON oder GEMINI (normal image)
— EPSON oder GEMINI (inverse image)
— CBM-1525/MPS-801 (oder Pendants dieser Drucker)
— EPSON mit Userport-Interface

Haben Sie Ihren Druckertyp ausgewählt, so wird zunächst ein weiteres Programm geladen.

Während Sie die ganzen Programme in den Speicher des C 64 holen, können sie die Maus zusammenbauen. Aus Gründen der Transportsicherheit ist die sogenannte Spurkugel noch nicht in der Maus enthalten. Auf der Unterseite ist eine Schraube zu lösen und die Spurkugel hineinzulegen.

In der Zwischenzeit dürfte auch das Hauptmenü zur Maussteuerung geladen sein. Nun können Sie die Maus einjustieren. Das Wie wird im Handbuch ausführlich beschrieben.

Abweichend von den meisten Mäusen weist die Magic-Maus drei Tasten in verschiedenen Farben (rot, blau, gelb) auf. Meist werden jedoch nur eine oder zwei der Tasten benötigt.

Im Hauptmenü können Sie bereits mit der Maus den gewünschten Punkt anwählen. Zur Verfügung stehen wie bereits erwähnt:

Zeichenprogramm für hochauflösende Grafik
Sprite-Entwurfsprogramm
Icon-Entwurfsprogramm (auch für den Zeichensatz)
und der Mouse-Controller

Bevor Sie anfangen, sollten Sie sich jedoch neben Ihrem Rechner mindestens einen Platz von 50x50 cm schaffen, auf dem die Maus ungehindert operieren kann.

Wenn Sie Ihren Menüpunkt angewählt haben — wir wählen zunächst einmal die hochauflösende Grafik — aktivieren Sie den gelben Knopf und das Grafikprogramm wird geladen.

7/1.8.1.1

Mitgelieferte Software

Der Preis der Magic-Maus wird nicht nur alleine durch die hardwaremäßige Maus gerechtfertigt, sondern auch durch die mitgelieferten Programme, die wir im Folgenden unter die Lupe nehmen möchten.

Zeichenprogramm für hochauflösende Grafik

Das Zeichenprogramm stellt grundsätzlich neun verschiedene Linienformen (BRUSHES genannt) zur Verfügung. Diese sind in dem anfangs sichtbaren Menüfeld dargestellt und können mit der Maus „angeklickt" werden. Im Menüfeld ist der Mauszeiger durch eine Hand mit ausgestrecktem Daumen und Zeigefinger dargestellt.

Generell dient im Grafikprogramm die gelbe Taste zum Anklicken der gewünschten Eigenschaften und die rote Taste zum Umschalten zwischen Menü und Zeichenblatt. Klicken Sie z.B. irgendeinen der Malstifte an, so verlöscht dieses Zeichen in dem vorgesehenen Feld und die Form des bisherigen Malstiftes (Voreinstellung: ein Punkt) erscheint wieder in seinem Feld. Am unteren Bildschirmrand ist ein Textfenster angegeben, bei dem Sie Ihre Texte eingeben können. Während des Zeichnens wird hier der ausgewählte Modus dargestellt.

Bei den Modi können Sie zwischen zwanzig unterschiedlichen Arten wählen, die wir im Folgenden kurz ausführen wollen:

Bezeichnung	Bedeutung	Vorgehensweise
DRAW	einfaches Zeichnen	Eine gedrückte gelbe Taste setzt einen Punkt in Form des gewählten BRUSHES an die Mausposition. Wird die Maus mit gedrückter Taste bewegt, so können auch Linien gezeichnet werden.
LINIE	Linie ziehen	Der Anfangspunkt wird zunächst mit Anklicken der gelben Taste markiert, dann der Grafikcursor auf den Endpunkt der Linie gebracht. Durch ein weiteres Anklicken der gelben Taste wird eine Linie zwischen diesen beiden Punkten gezogen.
LINES	fortlaufendes	Wie LINE, jedoch stellt der Endpunkt der einen Linie zugleich den Anfangspunkt der nächsten Linie dar.

Bezeichnung	Bedeutung	Vorgehensweise
FILL	Gebiet ausfüllen	Zunächst ist sicherzustellen, daß ein vollständig umgrenztes Gebiet ausgefüllt wird. Fehlt nur ein Bildschirmpunkt in der Eingrenzung, so wirkt der FILL-Befehl auch außerhalb der gewünschten Begrenzung, im schlechtesten Fall sogar auf den ganzen Bildschirm.
TEXT	Text in Grafik eintragen	Neben der Bezeichnung TEXT sind im Menüfeld noch die Begriffe SM und LA (abgeleitet von SMALL und LARGE) aufgezeigt. Durch Anklicken von SM erhalten Sie normale Schriftgröße und bei LA eine Breitschrift. Nachdem Sie wieder auf Ihr Zeichenbrett gewechselt haben, wird die Textausgabe mit der gelben Taste vorbereitet. Dabei kann man feststellen, daß der Cursor um einige Bildschirmzeilen springt. Damit sucht er sich die gültige Position für einen Text, da Text nur dort in die Grafik eingetragen werden kann, wo auch im Textmodus ein Zeichen plaziert würde. Frei wählbare Positionierung innerhalb der Grafik auf einzelne Bildschirmpunkte ist also nicht möglich.
FRAME	Rechtecke zeichnen	Zum Zeichnen des Rechteckes sind die linke obere und die rechte untere Ecke mit dem Grafikcursor anzuklicken.
BOX	ausgefüllte Rechtecke zeichnen	Wie bei FRAME
CIRCEL	Kreis zeichnen	Zunächst ist der Mittelpunkt des Kreises anzuklicken, anschließend irgend ein Radius.
RAYS	Strahlen zeichnen	Durch den Menüpunkt RAYS lassen sich Strahlen zeichnen, wobei die Vorgehensweise ähnlich wie bei LINES und CIRCEL ist. Zunächst wird ein Mittelpunkt der Strahlen angeklickt. Dieser Mittelpunkt ist dann Anfangspunkt aller Linien, die im Folgenden mit dem Cursor angeklickt werden.
AERO	Sprühen	Wie bei CIRCEL. Im definierten Umkreis werden mit AERO zufällig Bildschirmpixel mit dem gewählten Zeichenpinsel „gesprüht“.
CLEAR	Grafikbildschirm löschen	Sobald Sie nach Anwahl von CLEAR auf den Grafikbildschirm umschalten, wird dieser vollständig gelöscht.
ROB	Radieren	Dort wo sich der Grafikcursor befindet, werden durch Anklicken mit der gelben Taste alle Bildschirmpunkte gelöscht.

Bezeichnung	Bedeutung	Vorgehensweise
HORZ	horizontale Linie zeichnen	Nach Anklicken des Anfangspunktes einer Linie verläßt der Grafikcursor seine horizontale Position nicht mehr. Sofern Sie die Maus nicht horizontal bewegt haben, wird der Grafikcursor nach Zeichnen der Linie an die Position gesetzt, die Sie durch die normale Bewegung erreicht hätten. Dies ist immer ein Punkt in der Spalte, wo sich der Endpunkt der Linie befindet.
VERT	senkrechte Linie ziehen	Wie bei HORZ
FINE	Grafikcursor punktweise positionieren	Da die Maus nicht immer punktfein an einer gewünschten Position herangeführt werden kann, bietet das Zeichenprogramm die Möglichkeit, mit den Tasten Q, E, A, D, Z, X und C eine pixelfeine Bewegung durchzuführen.
LOAD	Laden von Grafiken	Im Textfenster wird der Name der Datei erfragt. Dateinamen dürfen dabei maximal 13 Zeichen lang sein, drei Beispiele befinden sich bereits auf der Diskette.
SAVE	Grafik speichern	Der Name wird im Textfenster erfragt.
PRINT	Hardcopy (normal)	
SHADE	Hardcopy ausdrucken	Falls Interface für Drucker am Userport angeschlossen.
EXIT	Zurück zum Hauptmenü	

Mit dem Zeichenprogramm lassen sich recht ansprechende Bilder darstellen, wie auch die drei auf Diskette befindlichen Beispiele (SAILING, MOON BASE und HATTER) zeigen. Wünschenswert wäre jedoch auch noch ein Zoom-Effekt gewesen, mit dem die Bildschirmpunkte auf 8x8 Punkte vergrößert werden. Dies würde dann ein saubereres Zeichnen ermöglichen, da auch Details besser zu gestalten sind.

Sprite-Entwurfsprogramm

Das Sprite-Entwurfsprogramm arbeitet naturgemäß mit einem Raster von 24 Spalten und 21 Zeilen. Zum Erstellen können die einzelnen Punkte in Textgröße gesetzt oder gelöscht werden. Sobald man aus dem Entwurfsrahmen heraus in die untere rechte Ecke wandert, erscheint ein Menü, das der Tastatur eines Taschenrechners nicht unähnlich sieht. Anhand des Menüs wollen wir die Möglichkeiten des Sprite-Entwurfsprogramms aufzeigen:

B	—	Hintergrundfarbe (alle Farben werden entsprechend ihrer Nummer durchlaufen)
C	—	Zeichenfarbe (siehe B)
0	—	Multicolor Sprite 0 (siehe B)
1	—	Multicolor Sprite 1 (siehe B)
U	—	Sprite-Zeigernummer um 10 erhöhen
D	—	Sprite-Zeigernummer um 10 vermindern
+	—	Nächstes Sprite auswählen
—	—	Vorhergehendes Sprite auswählen
M	—	Mehrfarbenmodus ein- und ausschalten
E	—	Aktuelles Sprite löschen
L	—	Spritedaten von Disk laden
S	—	Spritedaten auf Disk speichern
X	—	Sprite in X-Richtung dehnen
Y	—	Sprite in Y-Richtung dehnen
R	—	Sprite invers darstellen
Q	—	zurück ins Hauptmenü

Leider besteht keine Möglichkeit, die Sprites als Hardcopy in der Größe ihrer Erstellung auszudrucken. Ansonsten sind jegliche Hilfsmittel zur Erzeugung von Sprites gegeben.

Mehrfarbige Sprites können allerdings nicht mit ihrer Farbnummer eingegeben werden, sondern das Bitmuster ist in dem 21x24-Raster nachzuvollziehen. Hier wäre eine Farbwahl des aktuell zu zeichnenden Punktes über das Menü sinnvoll gewesen, da es trotz der Zeilen- und Spaltennumerierung nicht sehr einfach ist, mehrfarbige Sprites bitweise zu erstellen. Eine kleine Hilfestellung ist hier jedoch die Anzeige des Sprites in Originalgröße und Farbe, wenn man sowohl die X- als auch Y-Ausdehnung einschaltet. Vielleicht wäre es auch sinnvoll gewesen, die Punkte im Entwurfsblatt in der gewählten Farbe darzustellen. Ansonsten lassen sich Sprites sehr bequem erstellen.

Icons entwerfen/Zeichensatz ändern

Der Zeichensatz läßt sich auf ähnliche Weise wie die Sprites manipulieren. Je sechs Zeichen werden zu einer 2x3 Matrix zusammengefaßt. Aufeinanderfolgende Blöcke können zusammen verarbeitet werden und auch das Übernehmen des Zeichensatzes zum Ändern derselben ist möglich.

Besonders interessant ist diese Möglichkeit, da insgesamt 170 Icons generierbar sind, der Zeichengenerator des C 64 jedoch nur die ersten 86 Nummern benötigt.

Leider hat sich bei der Iconnummer eine kleine Unstimmigkeit eingeschlichen. Haben Sie z.B. eine dreistellige Iconnummer bereits angewählt gehabt, und gehen nun auf eine zweistellige Nummer zurück, so bleibt die dritte Ziffer erhalten.

Mauskontrolle

Wenn man die Magic-Maus besitzt, möchte man sicher nicht nur Sprites entwerfen, den Zeichensatz ändern oder Bilder „malen“. Interessant wird es, wenn man die Maus zur Steuerung eigener Programme einsetzen kann.

Dazu muß das Control-Programm geladen werden, was den Bereich von 52736 bis 53247 ($CE00 — $CFFF) belegt. Außerdem wird noch ein Teil des Kassettenpuffers für die Darstellung des Mauscursors benötigt, und zwar die Adressen 896 bis 959 ($0380 bis $03BF).

Dieser Mauscursor ist, wie Sie anhand der Adressen erraten, ein Sprite, das auch noch den Anwenderwünschen angepaßt werden kann.

Nachdem das Controlprogramm mit

```
LOAD „CONTROL“, 8, 1
```

geladen und mit

```
SYS 52800
```

gestartet wurde, kann man die Maus mittels Basic- oder Maschinenspracheprogramm kontrollieren oder den Mauscursor verändern.

Ein Basicprogramm wird durch die Maussteuerung selbst nicht beeinflußt bzw. das Bewegen der Maus auf dem Bildschirm wird dadurch nicht verlangsamt, da die Maus interruptgesteuert ist.

Um den Mauscursor vor jedem beliebigen Hintergrund erscheinen lassen zu können, kann man ihn dann z.B. mit POKE 600,1 bzw. POKE 600,0 blinken lassen, d.h. seine Farbe wird fortlaufend von weiß auf schwarz geändert und umgekehrt.

Mit POKE 53287 kann jede beliebige Farbe dem Mauscursor zugeordnet werden, sofern das Blinken abgeschaltet ist.

Am wichtigsten ist die Abfrage der Mausposition. Die aktuellen Koordinaten der Maus sind in den Adressen 53248, 53264 und 53249 abgelegt. Dies sind die Register der Koordinaten von Sprite 0, wobei zu berücksichtigen ist, daß die X-Koordinate nicht in einem Byte dargestellt werden kann. Das entsprechende Bit der Adresse 53264 (Register 16 des VIC) muß noch entsprechend ausgeblendet werden, wobei der Operator „AND 254“ verwendet wird.

In Register 0 des VIC (53248) ist die Basis-X-Koordinate zu erfragen und in Register 1 (53249) die Y-Koordinate. Etwas einfacher geht es, wenn man mit der Hälfte X-Koordinate zufrieden ist. In Adresse 593 ist dieser Wert abgelegt. Mit zwei multipliziert, ergibt der Wert in 50% aller Fälle das richtige Ergebnis, in den anderen Fällen liegt ein Fehler von einem Bildschirmpunkt vor. In Adresse 594 kann nochmals die Y-Koordinate abgefragt werden.

Bereits beim Zeichenprogramm hatten wir bei den Menüpunkten HORZ und VERT gesehen, daß die Maus nur in eine Richtung bewegt werden kann. Dafür ist Adresse 576 zuständig. Ist in diesem Byte eine 0 eingetragen, so kann sich die Maus frei sowohl in X- als auch in Y-Richtung bewegen. Eine 1 bedeutet, daß nur eine senkrechte Bewegung möglich ist, und eine 2 schränkt die Bewegungsfreiheit auf die Waagerechte ein.

Hier nochmals alle wichtigen Adressen der Maus in einer kleinen Tabelle:

Adresse	Bedeutung
576	Eingrenzen der Bewegungsrichtung
593	Halber Wert der X-Koordinate
594	Y-Koordinate
596	Abfragen der gelben Taste
600	Blinken ein- und ausschalten
52800	Startadresse für SYS
53216	Ausschalten des Maus-Controlers (SYS)
53248	Basis X-Koordinate
53249	Y-Koordinate
53264	MSB der X-Koordinate (Bit 0)

Mit diesen Kenntnissen wollen wir im Folgenden ein kleines Menüprogramm vorstellen.

7/1.8.1.2

Beispiele für die Mausverwendung

Nachdem wir nun die zum Lieferumfang gehörige Software besprochen haben, möchten wir einige Beispiele für die Anwendung der Magic-Maus in eigenen Basic-Programmen vorstellen.

7/1.8.1.2.1

Normales Menü

Um die Maus in einem Basic-Programm anwenden zu können, braucht man lediglich die Koordinaten der Mausposition zu erfragen. Als Beispiel haben wir für Sie ein Demo-Menü mit sechs Menüpunkten gewählt. Das Programm liegt in allgemeiner Form vor und kann den jeweiligen Wünschen entsprechend geändert bzw. in eigene Programme eingebaut werden.

```
MAUSMENUE

100 REM ------------------------------------------
110 REM ---            VORSPANN               ---
120 REM ------------------------------------------
130 :
140 IF A=0 THEN A=1 : LOAD"CONTROL",8,1
150 SYS52800
160 :
170 POKE 53280,6
180 POKE 53281,7
190 PRINT"[illegible]"
200 :
210 CD$="[illegible]"
220 CR$="[illegible]"
230 :
240 ZU$="[illegible]"
250 :
260 DIM TE$(6)
270 DIM ZE(6)
280 :
290 DATA "MENUEPUNKT 1",4
300 DATA "MENUEPUNKT 2",7
310 DATA "MENUEPUNKT 3",10
320 DATA "MENUEPUNKT 4",13
330 DATA "MENUEPUNKT 5",16
340 DATA "MENUEPUNKT 6",19
350 :
360 FOR I=1 TO 6
370 : READ TE$(I)
380 : READ ZE(I)
390 NEXT
400 :
410 S=12
420 :
430 RV$="[illegible]"
440 PRINT "[illegible]                    ------------------------------"
450 :
460 FOR MP=1 TO  6
470 : GOSUB 2000
480 NEXT
490 :
```

Listing 7/1.8.1.2.1 (1)

```
500 REM ------------------------------------
510 REM ----     MENUESCHLEIFE          ----
520 REM ------------------------------------
530 :
540 Y=PEEK(594)
550 X=2*PEEK(593)
560 :
570 IF X<120 OR  X>248 THEN NE=0 : GOTO 670
580 IF Y< 80 OR  Y>224 THEN NE=0 : GOTO 670
590 :
600 IF Y> 80 AND Y<104 THEN NE=1 : GOTO 670
610 IF Y>103 AND Y<128 THEN NE=2 : GOTO 670
620 IF Y>127 AND Y<152 THEN NE=3 : GOTO 670
630 IF Y>151 AND Y<176 THEN NE=4 : GOTO 670
640 IF Y>175 AND Y<200 THEN NE=5 : GOTO 670
650 IF Y>199 AND Y<224 THEN NE=6 : GOTO 670
660 :
670 IF NE=AL THEN GOTO 740
680 :
690 RV$="■" : MP=AL : GOSUB 2000
700 RV$="▨" : MP=NE : GOSUB 2000
710 :
720 AL=NE
730 :
740 IF PEEK(596)<>0 THEN GOTO 540
750 ON NE GOSUB 3100,3200,3300,3400,3500,3600
760 :
770 GOTO 430
780 :
790 END
800 :
1000 REM ------------------------------------
1010 REM ----   CURSOR POSITIONIEREN ----
1020 REM ------------------------------------
1030 :
1040 PRINT "▒";LEFT$(CD$,Z);LEFT$(CR$,S);
1050 :
1060 RETURN
1070 :
2000 REM ------------------------------------
2010 REM ---- MENUEPUNKT AUSGEBEN      ----
2020 REM ------------------------------------
2030 :
2040 IF MP=0 THEN RETURN
2050 :
2060 Z=ZE(MP)
2070 :
2080 GOSUB 1000
2090 :
2100 PRINT RV$;"|                    |";ZU$;
2110 PRINT RV$;"|  ";TE$(MP);"  |";ZU$;
2120 PRINT RV$;"L____________________|"
2130 :
2140 RETURN
2150 :
```

Listing 7/1.8.1.2.1 (2)

```
3100 REM ------------------------------
3110 REM ---- PROGRAMM ZU PUNKT 1     ----
3120 REM ------------------------------
3130 :
3140 PRINT"▮▮▮▮▮▮             PUNKT 1 WURDE ANGEWAEHLT"
3150 :
3160 FOR I=1 TO 1000 : NEXT
3170 :
3180 RETURN
3190 :
3200 REM ------------------------------
3210 REM ---- PROGRAMM ZU PUNKT 2     ----
3220 REM ------------------------------
3230 :
3240 PRINT"▮▮▮▮▮▮             PUNKT 2 WURDE ANGEWAEHLT"
3250 :
3260 FOR I=1 TO 1000 : NEXT
3270 :
3280 RETURN
3290 :
3300 REM ------------------------------
3310 REM ---- PROGRAMM ZU PUNKT 3     ----
3320 REM ------------------------------
3330 :
3340 PRINT"▮▮▮▮▮▮             PUNKT 3 WURDE ANGEWAEHLT"
3350 :
3360 FOR I=1 TO 1000 : NEXT
3370 :
3380 RETURN
3390 :
3400 REM ------------------------------
3410 REM ---- PROGRAMM ZU PUNKT 4     ----
3420 REM ------------------------------
3430 :
3440 PRINT"▮▮▮▮▮▮             PUNKT 4 WURDE ANGEWAEHLT"
3450 :
3460 FOR I=1 TO 1000 : NEXT
3470 :
3480 RETURN
3490 :
3500 REM ------------------------------
3510 REM ---- PROGRAMM ZU PUNKT 5     ----
3520 REM ------------------------------
3530 :
3540 PRINT"▮▮▮▮▮▮             PUNKT 5 WURDE ANGEWAEHLT"
3550 :
3560 FOR I=1 TO 1000 : NEXT
3570 :
3580 RETURN
3590 :
3600 REM ------------------------------
3610 REM ---- PROGRAMM ZU PUNKT 6     ----
3620 REM ------------------------------
3630 :
3640 PRINT"▮▮▮▮▮▮             PUNKT 6 WURDE ANGEWAEHLT"
3650 :
3660 FOR I=1 TO 1000 : NEXT
3670 :
3680 RETURN
```

Listing 7/1.8.1.2.1 (3)

Zunächst muß natürlich einmal das Mauskontrollprogramm geladen und mit SYS 52800 gestartet werden. Die beiden POKE-Befehle in den Zeilen 80 und 90 sowie der PRINT-Befehl in der Zeile 100 bestimmen die Farbgestaltung mit blauem Rahmen und blauer Schrift sowie gelbem Hintergrund.

Die beiden Variablen CD$ und CR$ dienen der Cursorsteuerung. Dazu sei hier auch auf das Unterprogramm ab Zeile 1000 hingewiesen. Die Variable ZU$ dient zum Setzen des Cursors bei der Menüausgabe. Da die Menüpunkte innerhalb einer Umrahmung ausgegeben werden, sind je Menüpunkt drei Zeilen auszugeben. ZU$ setzt den Cursor um die Länge der Menüausgabe zurück und eine Zeile tiefer.

Die Menüpunkte, die aktuell von der Maus angesteuert werden, sollen in inverser Schrift dargestellt werden. Dazu ist jeweils ein Umschalten zwischen inverser und normaler Schrift nötig. Der Einfachheithalber werden die Texte der Menüpunkte in einem Feld TE$() gespeichert und ZE() zeigt an, in welchen Zeilen am Bildschirm die Texte der Menüpunkte ausgegeben werden sollen. Haben Sie mehr oder weniger Menüpunkte, so sind die Feldgrößen in den Zeilen 170 und 180 entsprechend zu ändern.

Ab Zeile 200 wird zu jedem Menüpunkt der Text und die zugehörige Zeilennummer festgelegt und ab Zeile 270 diese Daten in die entsprechenden Felder eingelesen.

Die Spalte zur Textausgabe (Eingabeparameter für das Unterprogramm ab Zeile 1000 für die Spalte ist die Variable S) wird fest auf den Wert 12 gelegt.

Eine weitere Hilfsvariable bildet RV$. In ihr wird der jeweilige gewünschte Code für inverse Schrift ein/aus abgelegt. Zeile 350 druckt die oberste Zeile des Auswahlmenüs, und ab Zeile 370 wird das Unterprogramm zur Ausgabe einzelner Menüpunkte für jeden Menüpunkt aufgerufen. Da die Variable RV$ auf normaler Schrift steht, wird bei der Grundausgabe kein Menüpunkt invertiert.

Bevor wir zur Abfrage der Mausposition kommen, hier noch eine generelle Anmerkung: Da die Mausposition in Spritekoordinaten (Sprite 0) angegeben wird, ist jeweils ein sogenannter Offset von 24 Bildschirmpunkten in der X-Richtung und 50 Bildschirmpunkten in der Y-Richtung abzuziehen, wenn man die korrekte Position innerhalb des Bildschirmes erhalten möchte. Außerdem sind die Textzeichen jeweils mit 8x8 Bildschirmpunkten zu berechnen. Zur Abfrage reicht es bei uns, auf die Adressen 593 (halbe X-Koordinate) und 594 zurückzugreifen.

Zunächst werden in den Zeilen 570 und 580 alle Koordinaten abgefragt, die nicht innerhalb der Menüanzeige liegen. In diesem Falle wird der imaginäre Menüpunkt 0 ausgewählt. NE gibt jeweils den neuerrechneten Menüpunkt an. Ab Zeile 600 werden für die sechs Menüpunkte jeweils die Koordinaten erfragt und in NE der jeweilige Menüpunkt festgehalten.

In jedem Fall wird auf Zeile 670 übergegangen, wo abgefragt wird, ob überhaupt ein neuer Menüpunkt angewählt wurde. Wenn nicht, werden die folgenden Zeilen übersprungen und in Zeile 740 abgefragt, ob die gelbe Taste an der Maus gedrückt wurde. Wenn nicht, wird zur erneuten Abfrage der Mausposition übergegangen.

Andernfalls wird aufgrund des angewählten Menüpunktes zu den entsprechenden Programmteilen verzweigt, die in unserem Beispielprogramm ab Zeile 3100 stehen. Unsere Programmsequenzen beschränken sich dabei nur auf die Angaben des ausgewählten Menüpunktes und eine Warteschleife. Dies zur Kontrolle, ob das Programm richtig funktioniert.

Wird in Zeile 670 festgestellt, daß der neuangewählte Menüpunkt vom alten differiert, so wird zunächst in Zeile 690 mittels der Variablen RV$ auf normale Schrift umgeschaltet, und die Eingabevariable zur Ausgabe eines Menüpunktes (MP) auf den alten Menüpunkt eingestellt und sodann das erwähnte Unterprogramm aufgerufen. Anschließend wird dafür gesorgt, daß der neue Menüpunkt in inverser Schrift dargestellt wird. Auf jeden Fall wird noch in Zeile 720 dem Wert für den alten Menüpunkt das neue Ergebnis zugewiesen.

Cursor positionieren

Die Positionierung des Cursors wird unter Zuhilfenahme der Variablen CD$ (25x Cursor nach unten) und CR$ (40x Cursor nach rechts) bewerkstelligt. Zunächst wird der Cursor in die linke obere Ecke des Bildschirms gebracht, um einen definierten Ausgangspunkt zu erreichen. Dann wird von CD$ die gewünschte Anzahl von Zeichen abgetrennt und der gleiche Vorgang mit der Variablen CR$ wiederholt.

Zeile und Spalte bilden also die Eingabeparameter für das Unterprogramm zum Positionieren des Cursors. Wichtig ist das „;" am Ende von Zeile 1040, da sonst unser Unterprogramm unwirksam wäre.

Menüpunkt angeben

Eingabeparameter für das Unterprogramm zur Ausgabe eines Menüpunktes ist die Variable MP, die in den Zeilen 690 und 700 sowie als Laufvariable der Zeile 460 vorbesetzt wird. Aufgrund von MP wird zunächst die Zeile des Menüpunktes mit Hilfe von ZE() ermittelt und anschließend der Cursor positioniert.

Die Ausgabe des Menüpunktes erfolgt in drei Zeilen, wovon die mittlere den Text enthält. Aufgrund der Variablen RV$ wird der Menüpunkt entweder invers oder normal dargestellt. Die Funktion der Variablen ZU$ wurde bereits beschrieben.

7/1.11

Der C 64 als Oszilloskop

Autor: Andreas Frerichs

Das in *Bild 7/1.11-1* gezeigte Zusatzgerät, welches einfach auf den USER-Port des C 64 (C 128) gesteckt wird, verwandelt den Computer in ein digitales Oszilloskop, ein Speicheroszilloskop oder einen Oszillographen. Durch die verschiedenen Betriebsarten sind eine Vielzahl von Anwendungen möglich, die selbst mit Geräten der oberen Preisklasse nicht oder nur mit sehr aufwendigen Zusatzgeräten denkbar sind. Nach dem Einschalten und Starten des Programms zeigt sich das Titelbild des C 64 Oszilloskops *(Bild 7/1.11-2)*. Nachdem die Software vollständig geladen ist, zeigt sich der Leuchtschirm eines Oszilloskops. Auffällig ist nur, daß die horizontale Aufteilung in 16 Teile und nicht, wie üblich, in 10 Teile aufgeteilt ist.

Die Betriebsart Oszilloskop (DIREKT-Modus) ermöglicht es wie bei herkömmlichen Oszilloskopen periodische Spannungsänderungen, wie diese bei Sinus, Rechteck, Dreieck oder dem Brumm eines Netztransformators auftreten, darzustellen. Die Meßergebnisse werden direkt auf dem Bildschirm dargestellt.

Mit einem einzigen Knopfdruck wird der C 64 in ein komfortables Speicheroszilloskop (STORE-MODUS) umgewandelt. Maximal 10 Bildschirmseiten können anschließend betrachtet, gespeichert und ausgedruckt werden *(siehe Bild 7/1.11-3)*. Nun sind Messungen möglich, die mit normalen Oszilloskopen nicht realisierbar sind. Periodische und extrem langsame Spannungsänderungen, wie zum Beispiel eine abnehmende Schwingung, das Prellen eines Relais oder die Entladekurve eines Akkus können eindeutig sichtbar gemacht werden. Das Zusatzgerät erlaubt eine Meßwerterfassung bis zu 44,4 Stunden. Besonderes Bonbon dieser Hardware ist die Speicherung der Meßdaten auf Diskette um diese zu archivieren und zu einem späteren Zeitpunkt wieder abzurufen.

Wichtigster Bestandteil des C 64 Zusatzgerätes ist die mitgelieferte Software (auf Diskette). Diese ist äußerst komfortabel aufgebaut und nimmt auch Fehlbedienungen nicht übel.

Die Bedienanweisung erklärt auf 21 Seiten eindeutig und gut verständlich die Handhabung der Hardware. Hierbei fällt auf, daß eine untechnische Ausdrucksform gewählt wurde, so daß auch der Ungeübte schnell mit dem C 64 Oszilloskop umgehen kann.

Jedoch kennt auch diese Hardware ihre Grenzen. Wie bereits eingangs erwähnt, wird das Zusatzgerät auf den USER-Port gesteckt. Das bedeutet, der C 64 liest die Meßergebnisse nacheinander ein und speichert diese intern ab. Hier ergibt sich durch die Software eine Begrenzung bezüglich der Einlesefrequenz. Effektiv ergibt sich eine maximale Einlesefrequenz von 40 kHz, die allerdings nach dem sogenannten Abtast-Theorem mindestens um den Faktor 2 kleiner ist als Abtastfrequenz. Dadurch ergibt sich eine maximale Eingangsfrequenz von ca. 20 kHz. Ein Bereich, der, wie wir meinen, für den Hobby-Bastler völlig ausreichend ist.

Mit dem Zusatzgerät ist der C 64 wohl jedem, von Hobbyisten finanzierbaren Oszilloskop, überlegen. Auch der Bezugspreis, incl. Software, von DM 298,- stellt ein vernünftiges Preis/Leistungsverhältnis dar.

Bezug:
Microcomputer-Labor
Schumannstraße 23
6600 Saarbrücken

Technische Daten des C 64 Oszilloskop

Bildschirmdarstellung:	8x16 Linien, 10 Seiten Bildschirmspeicher
Y-Verstärker:	0,1; 0,2; 0,5; 1,0; 2,0; 5,0; 10; 20 und 50 Volt/Linie — calib.
Timebase:	12,5 μs/Linie bis 1000 s/Linie — calib.
Trigger:	intern oder extern über Taste
Meßgeschwindigkeit:	maximal 40 000 Messungen/Sekunde (theor.)
Meßdauer:	12,6 ms bis 44,5 h
Druckerroutinen:	Epson FX 80, MPS 801, MPS 803, CP 80X, Star SG 10 sowie alle Epson kompat.
Software:	5 1/4" Diskette

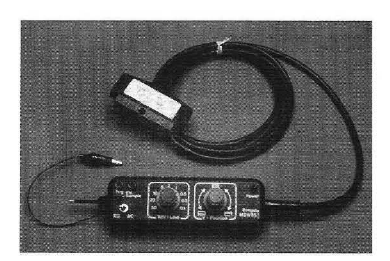


Bild 7/1.11 - 1: Das Zusatzgerät

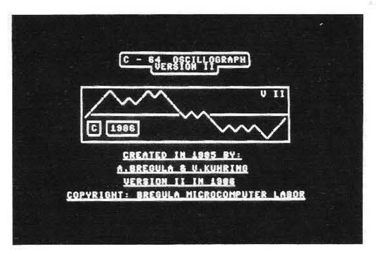


Bild 7/1.11 - 2: Das Titelbild



Bild 7/1.11 - 3: Das '64er-Oszilloskop' in der Anwendung

7/2
Bauanleitungen

2.1 Platinenlayouts

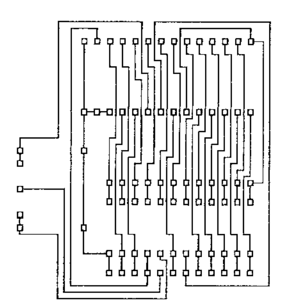


Bild 7/2.4–2
Platinenlayout (Leiterbahnseite)

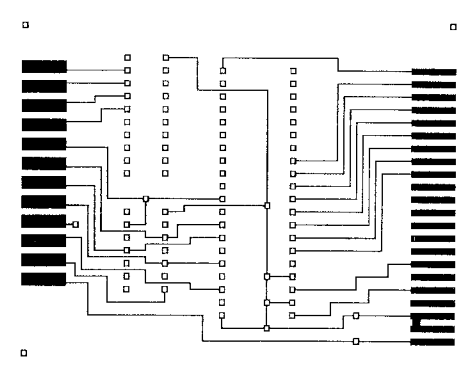


Bild 7/2.7–2a
Platinenlayout (Bestückungsseite)

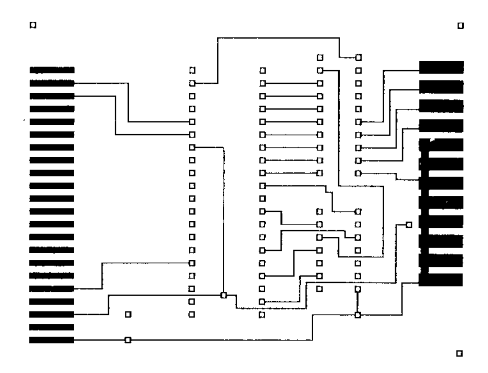


Bild 7/2.7–2b
Platinenlayout (Leiterbahnseite)

7/2.3

Druckerinterface (Centronics)

Autor: Dipl.-Inform. (FH) Rainer König

Wohl jeden ernsthaften Home-Computer-Besitzer befällt irgendwann einmal der Wunsch nach einem leistungsfähigen Drucker. Sei es, um selbstgeschriebene Programme endlich einmal schwarz auf weiß zu haben oder aber um Textverarbeitung zu machen oder ähnliches.

Nun bietet Commodore eine recht ansehnliche Palette an Druckern für ihre Home-Computer an, doch bei genauerem Betrachten der Geräte befällt so manchen Kauflustigen der Frust. Da fehlt einmal der Einzelblatteinzug, oder aber der Drucker ist recht langsam. Auch Umlaute sind nicht so ohne weiteres möglich. Am besten wäre wohl der Drucker der Marke XYZ (wir wollen hier keine Schleichwerbung machen), aber den gibt es nur mit Centronics-Anschluß, ein Interface zum seriellen Gerätebus des Commodore-Computers kostet dann noch extra und das nicht zu wenig.

Es geht aber auch anders. Mit der hier vorgestellten Lösung kann jeder Drucker mit Centronics-Schnittstelle an einen Commodore-64 angeschlossen werden. Die Kosten sind minimal, man muß nur das Kabel löten und die Software abtippen bzw. beim Verlag beziehen. Wer gar keine Löterfahrung hat, kann das Anschlußkabel auch in Computerfachgeschäften erhalten, aber der gar nicht schwierige Selbstbau ist die preiswerteste Lösung.

Welche Hardware wird benötigt?

Da der Drucker einen Centronics-Anschluß hat, braucht man natürlich ein geeignetes Verbindungskabel zum Computer. Dieses besteht aus zwei Steckern und einem 11poligen Kabel. Ein Stecker kommt an den Centronics-Anschluß des Druckers, der andere Stecker kommt auf den User-Port des Commodore-64, also nicht an den seriellen Bus, wo die Commodore-Drucker angeschlossen werden. Bild 7/2.3-1 zeigt, wie das notwendige Verbindungskabel auszusehen hat.

Achtung: Vor Einstecken des Kabels sind alle Verbindungen sorgfältig zu kontrollieren! Sowohl Rechner als auch Drucker müssen ausgeschaltet sein (das gilt auch für die Floppy!), da sie sonst Schaden nehmen können! Der Centronics-Stecker kann am Drucker nur auf eine Art eingesteckt werden, der User-Port-Stecker ist so am Rechner einzustecken, daß die Kabel auf der Unterseite angeschlossen sind! Am besten ist ein User-Port-Stecker mit Schutzkappe, auf der die Oberseite deutlich markiert ist.

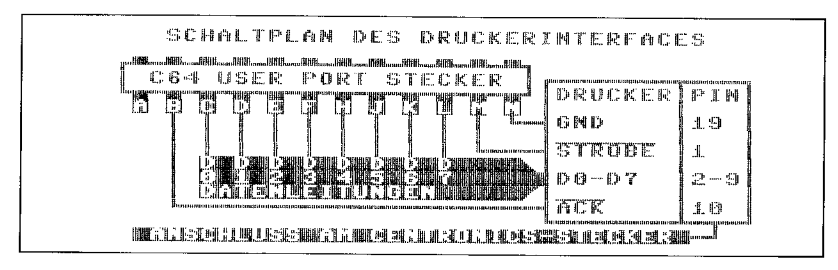


Bild 7/2.3-1 Das Anschlußkabel

Wie funktioniert die Schnittstelle?

Hier wollen wir kurz auf die Theorie der Schnittstelle eingehen.

Die Centronics-Schnittstelle ist so ungefähr das Einfachste, was man sich denken kann. Vom Rechner werden die 8 Datenleitungen zum Drucker verbunden, außerdem eine Strobeleitung, auf der der Rechner dem Drucker das Vorhandensein von Daten meldet. Der Drucker übernimmt die Daten vom Rechner und meldet diese Übernahme auf der Acknowledge-Leitung (ACK) zurück.

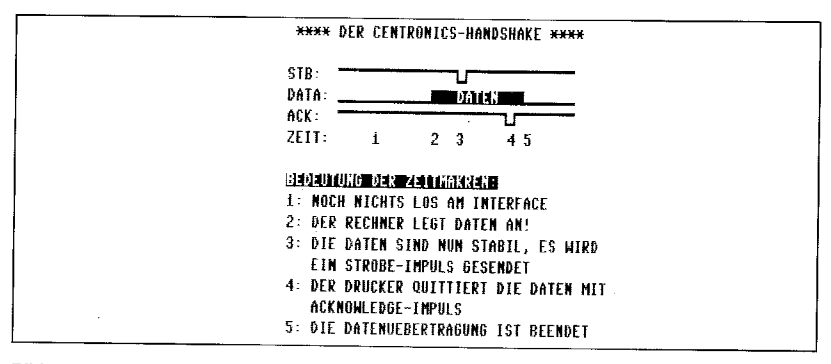


Bild 7/2.3-2 Ablauf einer Datenübertragung zum Drucker

```
5 OPEN4,4,13:CMD4
10 S$="‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|_|‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾"
20 D$="________________X   DATEN   ■________"
30 A$="‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|_|‾‾‾‾‾‾‾‾‾‾"
40 T$="     1         2  3       4 5       "
45 PRINTSPC(10);" **** DER CENTRONICS-HANDSHAKE ****"
50 PRINT:PRINTSPC(10);"STB:  ";S$
60 PRINTSPC(10);"DATA: ";D$
70 PRINTSPC(10);"ACK:  ";A$
80 PRINTSPC(10);"ZEIT: ";T$
85 PRINT:PRINTSPC(10);"BEDEUTUNG DER ZEITMARKEN:■"
90 PRINTSPC(10);"1: NOCH NICHTS LOS AM INTERFACE"
100 PRINTSPC(10);"2: DER RECHNER LEGT DATEN AN!"
110 PRINTSPC(10);"3: DIE DATEN SIND NUN STABIL, ES WIRD"
120 PRINTSPC(10);"   EIN STROBE-IMPULS GESENDET"
140 PRINTSPC(10);"4: DER DRUCKER QUITTIERT DIE DATEN MIT"
150 PRINTSPC(10);"   ACKNOWLEDGE-IMPULS"
160 PRINTSPC(10);"5: DIE DATENUEBERTRAGUNG IST BEENDET"
200 PRINT#4:CLOSE4
```

Bild 7/2.3-3: Beispiel für Druckerausgabe mit Centronics-Schnittstelle

Beim **Zeitpunkt 1** ist noch gar nichts auf der Druckerschnittstelle los. Man beachte auch gleich, daß die Leitungen STB (Strobe) und ACK (Acknowledge) mit negativer Logik arbeiten, also im Ruhezustand einen High-Pegel aufweisen.

Zum **Zeitpunkt 2** legt der Rechner nun Daten an den Ausgangsport. Es dauert nun einige Nanosekunden, bis die Daten stabil sind (wegen Signallaufzeiten, Leitungskapazitäten, etc.).

Zum **Zeitpunkt 3** signalisiert der Rechner dem Drucker, daß an den Datenleitungen gültige Daten anliegen. Hierzu wird die STB-Leitung kurzzeitig auf Low-Pegel gebracht, also ein STB-Impuls zum Drucker geschickt.

Der Drucker übernimmt daraufhin die anliegenden Daten und führt die entsprechenden Druckbefehle aus (z.B. Abspeichern im Puffer oder Ausdrucken).

Zum **Zeitpunkt 4** ist der Drucker mit der Verarbeitung der Daten fertig und meldet dies mit einem Impuls auf der ACK-Leitung. Danach wäre der Drucker für die Aufnahme neuer Daten wieder bereit.

Zum **Zeitpunkt 5** nimmt der Rechner die Daten wieder von der Leitung oder legt neue an, um das Spiel des Handshake zu wiederholen.

Diese Methode hat den Vorteil, daß sie einerseits recht schnell und andererseits sehr sicher ist.

Der einzige Nachteil für C-64-Benutzer ist, daß der Drucker den User-Port belegt, ein Telefonmodem kann also nicht gleichzeitig zum Drucker angeschlossen sein. Wenn man sich aber überlegt, daß einem mit dieser Lösung fast die gesamte

Druckerwelt offensteht, ist dieser kleine Nachteil durchaus verschmerzbar, zumal der User-Port relativ selten benutzt wird.

Viele im Handel angebotene Programme unterstützen diese Schnittstelle bereits, z.B. die Textverarbeitungen „Paper Clip“ und „Vizawrite“ sowie das Datenbankprogramm „Superbase 64“.

Wie sieht die Software aus?

Nachdem wir nun die Hardwarefragen geklärt haben, wollen wir die zum Betrieb der Schnittstelle notwendige Software vorstellen. Greifen wir uns doch gleich mal die zentrale Centronics-Ausgaberoutine aus dem Programmlistung heraus.

```
CENOUT: BIT  CIAICR       ;(1)
        STA  PDATA        ;(2)
        LDA  STBPORT      ;(3)
        AND  #STBO
        JSR  DELAY        ;(4)
        ORA  #STB1        ;(5)
        STA  STBPORT
        JSR  DELAY
WACK:   LDA  CIAICR       ;(6)
        AND  #%00010000
        BEQ  WACK
DELAY:  RTS
```

Zu (1): Hier wird das Interrupt-Flag-Register gelöscht. Wie aus dem Plan des Verbindungskabels (Bild 7/2.3-1) erkenntlich ist, wird die Acknowledge-Leitung am FLAG-Anschluß des User-Ports angeschlossen. Dieser ist mit Bit 4 des CIAICR verbunden. Die Maßnahme bezweckt nun, daß eventuelle Störungen auf dieser Leitung keinen Einfluß auf den Handshake haben, da hiermit eine definierte Ausgangssituation für die Datenübertragung geschaffen wird.

Zu (2): Der Rechner legt die im Akkumulator hinterlegten Daten an den User-Port an. Der vorherige BIT-Befehl hat zwar das CIAICR gelesen, aber den Akku dabei nicht verändert.

Zu (3): Das mit der Strobe-Leitung verbundene Bit des Ports A (PA2) wird mit dieser Sequenz auf Low-Pegel gezogen.

Zu (4): Das Programm ruft nun eine Zeitverzögerungsroutine auf. Somit hat der STB-Impuls eine genügend lange Dauer.

Zu (5): Die Strobe-Leitung erhält nun wieder ihren High-Pegel. Anschließend geht das Programm nochmals in die Zeitverzögerung.

Zu (6): Hier wird auf die Acknowledge-Meldung des Druckers gewartet. Solange das Bit 4 im CIAICR nicht gesetzt ist, kam auch noch kein ACK vom Drucker an, das Programm läuft also in der Schleife. Wenn durch einen ACK-Impuls das Bit gesetzt wird (es bleibt auch nach Wegfall des Impulses gesetzt), dann läuft die Ausgaberoutine auf ihr Ende.

Zu (7): Dies ist die Beendigung der Ausgaberoutine. Um ein Byte zum Drucker zu senden, muß es also nur im Akku hinterlegt werden (wird zerstört) und sodann ein JSR cenout gegeben werden. Zugleich ist der RTS-Befehl die Zeitverzögerungsroutine, denn ein JSR. . .RTS benötigt ja auch immerhin ein paar Taktzyklen, genug, um klare Impulse für den Drucker zu generieren.

Somit ist bereits eine minimale Centronics-Treibersoftware geschrieben. Allerdings geht es leider nicht ganz so einfach. Folgende Dinge müssen mindestens noch beachtet werden:

- Um den Drucker genauso über OPEN. .CLOSE wie einen Commodore-Drucker ansprechen zu können, muß die Ausgaberoutine ins Betriebssystem eingebracht werden.
- Commodore hält sich leider nicht an den ASCII-Code, den der Drucker versteht. Um keinen uferlosen Zeichensalat auf dem Drucker zu bekommen, müssen die Zeichen gegebenenfalls von Commodore-ASCII nach Drucker-ASCII umgewandelt werden.
- Die wenigsten Drucker können die besonderen Steuerzeichen des Commodore (z.B. für Cursorsteuerung) drucken. Hier muß eventuell auf eine Hilfsdarstellung ausgewichen werden.

Was kann unsere Druckersoftware?

Hier wollen wir kurz umreißen, was die von uns entwickelte Treibersoftware alles kann. Dadurch wird vielleicht auch klar, warum aus ein paar Zeilen Assembler plötzlich 1 kByte-Programm wurde.

- Alle Funktionen sind über Sekundäradressen von 0..15 steuerbar.
- Beliebige Zuordnung der Funktionen zu den Sekundäradressen.
- Zeichen können direkt (unverändert) zum Drucker geschickt werden.
- Umwandlung von Commodore-ASCII nach Drucker-ASCII.
- Wahlweises Setzen des höchsten Datenbits, dies schaltet bei einigen Druckern Kursivschrift ein.
- Graphikfähige Drucker können den gesamten Commodore-Zeichensatz drucken, er wird aus dem Character-ROM des C 64 entnommen.
- Besondere Behandlung der Steuerzeichen:
  — Unterdrücken der Ausgabe
  — Durch SPACE ersetzen
  — Als ASCII-Zeichen in Breitschrift drucken
  — Als Commodore-Zeichen im Graphikmodus drucken
- Hardcopies können erstellt werden von:
  — Textbildschirm
  — Hires-Grafikbild, egal wo im Speicher
  — Multicolorbild, egal wo, wird mit 4 Graustufen gedruckt

Bild 7/2.3-1 und Bild 7/2.3-2 sowie das Listing in Bild 7/2.3-3 (welches Bild 7/2.3-3 druckt) wurden natürlich mit unserer Treibersoftware erstellt. Bild 7/2.3-1 zeigt zudem recht eindrucksvoll, daß ein Hardcopy vom Textschirm nun zum Kinderspiel wird, auch Graphik-Hardcopies sind mit einem einzigen PRINT-Befehl zu erzeugen.

Nun wollen wir ganz speziell auf die Treibersoftware eingehen, allerdings wollen wir nicht nochmals jede Zeile breittreten. Dies kann den zahlreichen Kommentaren im Programmlisting einfacher entnommen werden. Vielmehr wollen wir hier zeigen, welche besonderen Programmiertricks angewandt wurden, um soviel Druckleistung in so wenig Speicher zu packen. Auf diese Art kann der Leser wahrscheinlich mehr aus dem Programm lernen und erhält vielleicht auch richtungsweisende Impulse für seine zukünftigen Eigenentwicklungen. So wird bei der Multicolor-Hardcopy beispielsweise ein Zeiger auf einen Zeiger, der wiederum auf einen Zeiger zeigt, verwendet. Das hört sich zwar irre verrückt an, ist jedoch eigentlich relativ einfach. Doch nun der Reihe nach . . .

Wie wurde das Programm ins Betriebssystem eingebunden?

Eine der wichtigsten Funktionen einer Software-Druckerschnittstelle ist, daß sie dem Benutzer nicht als Software-Lösung erscheint, d.h. der Benutzer soll gar nicht merken, daß kein normaler Commodore-Drucker angeschlossen ist. Die Schnittstelle muß wie bei CBM-Druckern mit OPEN, CLOSE und PRINT# anzusprechen sein, nur so ist gewährleistet, daß Programme nicht extra für die Schnittstelle umgeschrieben werden müssen.

Um dies zu erreichen, werden von unserer Software drei Vektoren verändert, die auf Kernal-Routinen zeigen.

CHKVEC wird bei OPEN-Befehlen angesprungen. Die Änderung dieses Vektors ist erforderlich, da bei OPEN-Kommandos mit Angabe einer Sekundäradresse bereits der serielle Bus aktiviert wird. Wird dies nicht vom Software-Interface abgefangen, so führt ein OPEN zu einem „?DEVICE NOT PRESENT ERROR“.

OUTVEC wird bei PRINT#-Befehlen angesprungen. Der Akku enthält das auszudruckende Zeichen, unsere Software bereitet es eventuell noch für den Druck auf. Um festzustellen, ob es überhaupt für den Drucker bestimmt ist, wird die gerade aktive Geräteadresse (CURDEV) mit der im RAM hinterlegten Adresse unseres Interfaces (PRIDEV, Standardwert 4) verglichen. Ist der Drucker gemeint, so wird die Ausgabe von unserer Routine durchgeführt, ansonsten übernimmt die Kontrolle wieder das Kernal-ROM.

NMIVEC wird auf eine kleine Routine gelegt, die beim Drücken von RUN/STOP-RESTORE das Druckerinterface wieder initialisiert.

Wie erfolgt die Steuerung der Schnittstellenfunktionen?

Um das Interface möglichst flexibel zu gestalten, erfolgt die Steuerung der Funktionen über die Sekundäradressen 0..15. Hierbei ist zu beachten, daß die Zuordnung

einer Funktion zu einer Sekundäradresse nicht fest ist, sondern flexibel über eine Tabelle gelöst wird. Die Auswertung der Sekundäradresse geschieht in der Routine NEWCHK, hier wird die aktuelle Sekundäradresse ausgewertet und das entsprechende Funktionsbyte aus der Tabelle in der Speicherstelle FUNCTION hinterlegt.

Die Bits im Funktionsbyte haben folgende Bedeutung:

7 Das Setzen von Bit 7 veranlaßt das Programm, Hardcopies zu drucken. Diese Funktion hat Priorität vor allen anderen.

6 Das Setzen von Bit 6 führt dazu, daß alle Zeichen ohne jegliche Umsetzung durch unser Programm zum Drucker geschickt werden. Diese Funktion hat wiederum Priorität vor allen folgenden.

5 Ist Bit 5 gesetzt, so wird ein Commodore-Drucker simuliert, d.h. alle Zeichen werden im Graphikmodus des Druckers gedruckt, die Zeichenmatrix wird dabei aus dem Character-ROM des Rechners gelesen (in Abhängigkeit von Bit 2). Mit dieser Funktion lassen sich sehr gut Programmlistings erstellen.

4 Das Setzen von Bit 4 veranlaßt das Interface, bei der Ausgabe von Zeichen an den Drucker bei diesen das höchstwertige Bit (Bit 7) zu setzen. Bei einigen Druckern schaltet dies Kursivschrift ein.

3 Das Setzen von Bit 3 aktiviert die Vertauschung von Groß- und Kleinbuchstaben.

2 Mit diesem Bit wird dem Interface bekanntgegeben, welcher Zeichensatz des Character-ROMs beim Graphikdruck verwendet werden soll.

0, 1 Mit diesen beiden Bits wird angegeben, wie bei Control-Zeichen (z.B. CLR/HOME) verfahren werden soll:

00 Unterdrücken
01 als Leerzeichen drucken
10 in Breitschrift als ASCII-Zeichen drucken
11 Zeichen aus Character-ROM holen

Durch eine entsprechende Sekundäradressentabelle kann also jede Funktion einer beliebigen Sekundäradresse zugeordnet werden. Das Interface-Programm ist somit leicht an bestehende Programme und deren Druckfunktionen anpaßbar. Im Listing ist die Sekundäradressentabelle so gewählt, daß das Programm zu einem bereits bestehenden Software-Interface einer Göttinger Firma weitgehend kompatibel ist.

Was geschieht mit den Zeichen?

Hier wollen wir uns ansehen, was unsere Druckersoftware mit den Zeichen anstellt, wenn wir diese nicht direkt zum Drucker schicken wollen bzw. wenn wir keine Hardcopy-Funktion auslösen wollen.

Zunächst sei hier erwähnt, daß das Interface-Programm zwei Speicherstellen kennt, in denen es Informationen über den Betriebsmodus hinterlegt:

QUOTFLG ist ungleich Null, wenn das auszugebende Zeichen hinter einem Anführungszeichen steht. Dieses Flag wird von CR und LF zurückgesetzt und dient dazu, Commodore-Kontrollzeichen zu identifizieren (z.B. PRINT "CLR/HOME").

RVSFLG wird von den Steuerzeichen (RVS-ON) bzw. (RVS-OFF) gesetzt oder zurückgesetzt.

Die Auswertung auf Anführungszeichen, RVS-Steuerzeichen und CR (Wagenrücklauf) bzw. LF (Zeilenvorschub) erfolgt auf jeden Fall.

Ab dem Label CONVERT erfolgt nun gegebenenfalls eine Wandlung der Zeichen.

So mancher Leser wird nun wissen wollen, warum die Zeichen umgewandelt werden müssen. Dazu einige Anmerkungen zur Politik von Commodore, bezüglich ihres eigenen Industriestandardes:

Das, was Commodore als ASCII-Zeichensatz bezeichnet, hat mit dem von ANSI (American National Standard Institute) genormten ASCII-Satz (den die Drucker verstehen) nicht viel gemeinsam. Zum einen sind die Steuerzeichen weitgehend inkompatibel, Groß- und Kleinbuchstaben sind vertauscht, manche Zeichen sind sogar doppelt vorhanden. Wer es nicht glaubt, kann folgende BASIC-Zeile eingeben:

```
PRINT CHR$(97);CHR$(193)
```

Wir sehen zwei gleiche Zeichen am Schirm auftauchen, obwohl wir unterschiedliche (Commodore-Pseudo-)ASCII-Werte als Argumente angegeben haben. Dies ist auch der Grund, warum manche Textverarbeitungsprogramme beim Lesen von sogenannten ASCII-Files Schwierigkeiten haben. Historisch gesehen, ist dies auf den „Ur-C 64", den PET 2001, zurückzuführen, bei dem u.a. mit Shift die Kleinbuchstaben erzeugt wurden. Dies erleichterte damals (1979/80) die Programmierung.

Dieses Problem der Zeichenumsetzung in ein Format, welches die üblichen Drucker verstehen, kann man ja gottseidank durch schlaue Programmierung in den Griff bekommen. Aber wie geht man vor?

Eine Möglichkeit wäre eine Umsetzungstabelle, hier wird jedem CBM-ASCII-Zeichen ein Druckerzeichen zugeordnet. Dummerweise belegt eine solche Tabelle 256 Byte, und durch die eventuell notwendige Vertauschung von Groß- und Kleinschrift ist eine zweite Tabelle erforderlich, das Programm zur Umsetzung würde also zu groß werden.

Die nächste Möglichkeit ist, je nach Zeichencode bestimmte Werte zu den Commodore-Codes hinzuzuzählen bzw. abzuziehen. Das kann in viele Abfragen ausufern.

Nun wollen wir die Methode vorstellen, welche wir als optimal ansehen. Sie ist zwar etwas unorthodox, aber belegt minimalen Speicherplatz.

Die Zeichen werden zweistufig umgesetzt. Zuerst erfolgt eine Umsetzung von CBM-ASCII-Format in einen Bildschirm-Code. Dies hat folgende Vorteile:

- Commodore-Drucker lassen sich sehr leicht realisieren, da ein derartig umgewandeltes Zeichen direkt als Zeiger in das Character-ROM des Computers verwendet werden kann.
- Die mehrfachen Code-Bereiche werden eliminiert. Control-Zeichen sind zudem am gesetzten Bit 7 des Bildschirmcodes zu erkennen.

Die Umsetzung vom CBM-ASCII-Wert in einen Bildschirmcode ist relativ einfach. Eigentlich sollte dies zuerst mit einer ROM-Routine erreicht werden (das Kernal muß das ja auch irgendwie schaffen), aber die ROM-Routinen sind dermaßen haarsträubend programmiert und zudem nicht von außen zugänglich, daß nichts anderes übrigblieb, als das Rad der Umwandlung zweimal zu erfinden.

Sieht man sich die Code-Bereiche an, so erkennt man, daß das Hinzuzählen bzw. Abziehen von Werten der richtige Weg ist. Zudem ist es erfreulich, daß in jeweils 32-Byte-Feldern immer die selben Werte addiert bzw. subtrahiert werden müssen. Die einzige Ausnahme ist der CBM-ASCII-Code 255, er muß in den Bildschirmcode 94 umgesetzt werden, aber das ist auch noch zu schaffen. Ansonsten gilt folgende Tabelle:

Codebereich	Versatz	neuer Bereich
0 — 31	+128	128 — 159
32 — 63	0	32 — 63
64 — 95	—64	0 — 31
96 — 128	—32	64 — 95
128 — 159	+64	192 — 223
160 — 191	—64	96 — 127
192 — 223	—128	64 — 95
224 — 255	—128	96 — 127

Nun sind Subtraktionen auch nur Additionen mit negativem Vorzeichen. Bei byteweiser Addition — wie hier vorgenommen — kann man einen SBC #64 auch durch einen ADC #192 ausdrücken.

Nun wird es kinderleicht. Das ursprüngliche Zeichen wird um 5 Bit nach rechts verschoben (Division durch 32). Man erhält einen Zeiger in eine 8 Byte lange TABELLE, hier ist hinterlegt, welches Byte hinzuaddiert werden muß, um auf den Bildschirmcode zu kommen. Somit ist der erste Umwandlungsschritt mit einer kleinen Tabelle und ein paar Programmzeilen zu bewerkstelligen.

Der gegebenenfalls erforderliche zweite Umwandlungsschritt, die Umsetzung von Bildschirmcode in ein Drucker-ASCII-Zeichen (Routine STOASC), geschieht fast genauso. Hier interessieren jedoch nur die unteren 7 Bits des Zeichens. Die Umsetzungstabelle ist ähnlich aufgebaut, sie hat auch 8 Byte, obwohl hier eigentlich nur

4 Byte erforderlich wären. Aber hier wird auch die unter Umständen erforderliche Vertauschung von Klein- und Großbuchstaben vorgenommen, d.h. es werden eigentlich zwei Tabellen zu je 4 Byte verwendet.

Man kann an diesem sehr realen Beispiel eigentlich sehr gut erkennen, daß der naheliegendste Weg (Umsetzung über 256-Byte-Tabelle) nicht immer der optimale ist. Die hier vorgestellte Lösung erforderte zwar etwas mehr Gehirnschmalz, ist aber auf jeden Fall günstiger.

Wie werden Commodore-Graphikzeichen gedruckt?

Die Beantwortung dieser Frage ist sehr einfach. Wie wir gesehen haben, werden die Zeichen zweistufig umgesetzt. Nach der ersten Umsetzung erhalten wir die Nummer des Commodore-Zeichens im Character-ROM. Mit Hilfe des Bits 2 im Funktionsbyte wird nun ein Zeichensatz ausgewählt. Die Routine READROM liest nun die Zeichenmatrix aus dem Character-ROM und speichert sie im Feld Matrix.

Mit der Routine ILOOP wird eine ESC-Sequenz zum Drucker geschickt, die diesen in den Graphikmodus umschaltet, d.h. mit den nächsten 8 Zeichen werden die Nadeln des Druckers angesteuert. Um die Routine so flexibel wie möglich zu gestalten, werden die hierzu notwendigen Steuercodes dem 4 Byte langen Feld GMTABLE entnommen. Inhalt:

```
ESC+"L"+CHR$(8)+CHR$(0)
```

Die nachfolgende MATOUT-Routine gibt nun das Feld MATRIX zum Drucker aus. Hierbei werden zuerst die 8höchstwertigen Bits ausgegeben, dann die nächste Bitspalte usw. Das MATRIX-Feld wird also um 90 Grad gedreht.

Die Routine MATOUT wird auch von den Hardcopy-Routinen benutzt, die wir uns nun ansehen wollen.

Wie funktionieren die Hardcopy-Routinen?

Folgendes ist allen drei Hardcopy-Arten gemeinsam:

- Der Zeilenabstand wird auf 8/72 Zoll gesetzt, dies entspricht bei den meisten Druckern der Höhe der Druckmatrix.
- Vor jeder Hardcopyzeile wird der Drucker auf Einzelnadelbetrieb gesetzt, wobei nun eine ganze Zeile (320 bzw. 640 Punkte) übergeben wird (mit Routine MATOUT).
- Während des Druckens wird anhand der File-Nummer unterschieden, ob zusätzlich zu einem Wagenrücklauf ein Zeilenvorschub geschickt werden muß (Filenummer > 127).
- Nach Beendigung der Hardcopy wird der Drucker wieder auf 6 Zeilen/Zoll Zeilenabstand für normale Textausgabe gesetzt.

Die Texthardcopy

Diese Funktion ist sehr einfach. In der Speicherstelle 648 ist hinterlegt, bei welcher Speicherseite der Bildschirmspeicher anfängt. Die nächsten 1000 Byte ab dieser Adresse sind der Bildschirminhalt, er kann nun im Prinzip genauso ausgegeben werden wie Commodore-Graphikzeichen.

Die Hires-Hardcopy

Diese Funktion ist nicht ganz so einfach. Der Benutzer muß hierbei die Nummer des 8-KByte-Blocks übergeben, in dem die Bitmatrix startet. Die meisten Graphikerweiterungen legen ihre Bilder ab $E000 ab, d.h. es muß eine „7“ zum Interface ausgegeben werden.

Nun werden 1000 Felder zu je 8 Byte in die MATRIX eingelesen und zum Drucker geschickt. Hierbei ist zu beachten, daß sämtliche ROMs desaktiviert werden, wenn das Programm auf die Graphikbytes zugreift, denn diese können ja unter den ROMs liegen.

Die Multi-Color-Hardcopy

Diese Funktion wollen wir nun näher betrachten. Zunächst sehen wir uns an, wie sie aufgerufen wird. Der Benutzer schickt in diesem Fall zum Funktionskanal des Interfaceprogramms folgenden ASCII-Code:

128+Nummer der 1-K-Seite der Farbinformation

Die 128 ist gewählt, damit das Interface beim Empfang von CR (CHR$(13)) oder ähnlichen „normalen“ Zeichen nicht gleich anfängt, unsinnige Hardcopies zu produzieren. Zu dieser 128 wird nun die Nummer der Seite des Farb-RAMs addiert. Damit hat es folgende Bewandtnis:

Kennt man die Lage des Farb-RAMs (hiermit ist der 1-K-Bereich gemeint, der normalerweise als Bildschirmspeicher fungiert, bei Hardcopy-Bildern steht seine Adresse leider nicht immer in 648), so kennt man eigentlich schon alle Adressen für das Multicolorbild:

- Die Hintergrundfarbe steht grundsätzlich in 53281.
- Die Farben, die durch die Bitkombinationen 01 und 10 adressiert werden, liegen in besagtem Farb-RAM.
- Die letzte Farbe (Bitkombination 11) liegt im Standard-Farb-RAM ab 55296.

Um nun die Adresse der Bitmatrix zu ermitteln, geht man von folgender Gesetzmäßigkeit aus:

- Der Videocontroller kann nur 16 KByte addressieren.
- Daraus folgt, daß die Bitmatrix in derselben 16-K-Bank liegen muß wie das Farb-RAM.

- Da das Farb-RAM aber nicht im selben Bereich liegt wie die Bitmatrix, nimmt man als Adresse der Bitmatrix einfach die Adresse des 8-K-Blocks, in dem die Farbinformation **nicht** steht.

Zugegeben, das hört sich etwas verwirrend an, daher hier ein kurzes Beispiel:

Der an das Interface übergebene Wert sei 132, also 128+4.

Die Farbinformation liegt also in den Adressen ab 1024, das ist im ersten 16-KByte-Block. Da die Bitmatrix der Graphik nicht die Adressen des Farb-RAMs nutzen kann, muß diese Matrix also logischerweise bei 8192 beginnen, also dem zweiten 8-K-Block im ersten 16-K-Block.

Genau diese Rechenoperationen führt unser Programm auch durch, wobei dies an manchen Stellen geradezu in Bit-Fuchserei ausartet.

Nun wollen wir beschreiben, wie die Graustufen realisiert wurden. Hierbei wurde der Drucker auf doppelte Nadeldichte gesetzt, 320-Graphikpunkte entsprechen also 640 Druckpunkten. Da zwei Graphikpunkte sowieso einer Graustufe entsprechen, stehen 4 Druckpunkte für die Realisation einer Graustufe zur Verfügung.

Die vier Graustufen sind nun mit folgenden Druckbildern realisiert:

weiß	0000	keine Nadeln
hellgrau	0011	
dunkelgrau	1010	
Schwarz	1111	alle Nadeln.

Um nun zu vermeiden, daß beim Drucken von grauen Flächen senkrechte Linien entstehen, werden die Bits für die Graustufen (0011 und 1010) in jeder ungeraden Graphik-Zeile invertiert. Eine dunkelgraue Fläche kann also folgendermaßen gedruckt werden:

Gerade	Zeilennummer:	1010101010101010101010101010
Ungerade	Zeilennummer:	0101010101010101010101010101
Gerade	Zeilennummer:	1010101010101010101010101010
Ungerade	Zeilennummer:	0101010101010101010101010101

Im Programm wird diese Funktion dadurch realisiert, daß die Tabelle der Graustufendarstellung (GREYIMGS) 8 Byte groß ist, bei geraden Zeilennummern wird auf die ersten 4 Bytes zugegriffen, bei ungeraden Zeilennummern auf die letzten 4 Bytes.

Die Ermittlung der Graustufe zum Drucken geschieht in folgenden Schritten:

1. Für jedes 8Byte große Graphikquadrat wird die Farbinformation ermittelt, die Methode wurde weiter oben schon beschrieben.
2. Die 8 Graphikbytes werden nach MATRIX gebracht.
3. Nun werden die jeweils beiden höchsten Bits der Bytes in MATRIX herausgeschoben, sie bilden einen Zeiger auf die Tabelle, welche die für das gerade bearbeitete Quadrat ermittelten Farben enthält.

4. Der dort stehende Wert wird wiederum als Zeiger in eine Tabelle der Farbe-Graustufen-Zuordnung (GREYTBL) verwendet.
5. Hier steht nun ein Zeiger auf die Graustufen-Druckbild-Tabelle (GREYIMGS).
6. Dieser Zeiger wird noch mit der Information verknüpft, ob gerade eine ungerade Zeile bearbeitet wird, in diesem Fall wird 4 zum Zeiger addiert.
7. Das nun adressierte Graustufendruckbild wird an die richtige Stelle der Matrix GREYMAT geschoben.
8. Sind alle 8 Bytes der ursprünglichen Matrix bearbeitet, so wird GREYMAT ausgedruckt.

Um irgendwelche Farben anderen Graustufen zuzuordnen, muß die Tabelle GREYTBL entsprechend angepaßt werden.

Somit sind wir am Ende der Programmbeschreibung angelangt, nun wollen wir noch kurz einen Blick auf die Handhabung werfen.

Wie wird das Programm bedient?

Hier wird vorausgesetzt, daß das Programm als Objektprogramm (Maschinenprogramm) auf Diskette vorliegt, und zwar unter dem Namen „PRINTERFACE".

Das Programm wird nun wie folgt gestartet:

```
LOAD „PRINTERFACE",8,1
SEARCHING FOR PRINTERFACE
LOADING
READY.
NEW
READY.
SYS 49152
```

Der NEW-Befehl ist erforderlich, da das Laden von Maschinenprogrammen die BASIC-Zeiger durcheinanderbringt, mit NEW werden für BASIC wieder klare Verhältnisse geschaffen.

Selbstverständlich können Sie das Programm auch für andere Adreßbereiche assemblieren, nötigenfalls diese aber dann gegen Überschreibungen durch BASIC schützen. Die Startadresse beim SYS-Befehl muß dann auch entsprechend geändert werden.

Nun können Sie Ihren am User-Port angeschlossenen Drucker mit ganz normalen OPEN-, PRINT#, CMD- und CLOSE-Befehlen bedienen.

Die Geräteadresse des Druckers ist 4, dies kann durch Ändern des Bytes PRIDEV jedoch auch angepaßt werden.

Die Funktionen des Interfaces werden durch die Sekundäradressen 0..15 gesteuert, ihre Zuordnung steht in der Tabelle FUNTBL und kann natürlich an spezielle Bedürfnisse angepaßt werden.

Zum Schluß noch ein Tip: Sollte Ihr Drucker nur permanent auf einer Zeile drucken, so sehen Sie einmal im Drucker-Handbuch nach. In fast jedem Drucker gibt es einen kleinen Mikroschalter, mit dem man ihn auf Auto-Line-Feed stellen kann, d.h. bei jedem CR, das der Drucker empfängt, führt er auch einen Zeilenvorschub aus. Sollten Sie keinen solchen Schalter finden, dann tut es auch eine Filenummer größer 127 (z.B. OPEN 132,4,0), der C 64 schickt dann mit jedem CR ein LF-Zeichen.

Nun bleibt uns nur noch, Ihnen viel Spaß mit dieser leistungsfähigen Erweiterung zu wünschen.

```
;**********************************
;* CENTRONICS-INTERFACE AUF C64  *
;*-------------------------------*
;* AUTOR: RAINER KOENIG          *
;*-------------------------------*
;* VERSION 2.1 VOM 20.05.1986    *
;**********************************
;
                .OS
                .BA $C000          ;START ASSEMBLY
;**********************************
;* EXTERNE ADRESSDEKLARATIONEN   *
;**********************************
;
;SYSTEMVARIABLE IN DER ZERO-PAGE
;==============================
;
CURDEV:         .EQ $9A            ;CURRENT DEVICE#
CURSEC:         .EQ $B9            ;CURRENT SEC-ADRESS
;
;ZEIGER IN DER ZERO-PAGE
;======================
;
ROMPTR:         .EQ $FB            ;-> ZEICHENMATRIX
SCRPTR:         .EQ $FD            ;-> VIDEO-RAM
CLRPTR:         .EQ $FD            ;-> COLOR-RAM
;
;KERNAL-SYSTEMVEKTOREN IM RAM
;===========================
;
CHKVEC:         .EQ 800            ;CHKOUT-VEKTOR
OUTVEC:         .EQ 806            ;CHROUT-VECTOR
```

Listing 7/2.3-1 Teil 1

```
NMIVEC:         .EQ 792             ;NMI-VECTOR
;
;USER-PORT-CIA REGISTER
;=====================
;
STBPORT:        .EQ $DD00           ;PORT FUER STROBE
PDATA:          .EQ $DD01           ;DATENPORT CENTRONIS
STBDIR:         .EQ $DD02           ;DATENRICHTUNG STB
DATADIR:        .EQ $DD03           ;RICHTUNG B
CIAICR:         .EQ $DD0D           ;INTERRUPTCONTROL
;
;DEFINITION VON KONSTANTEN
;=========================
;
IMASK:          .EQ %10000000       ;INTERRUPTMASKE
STB1:           .EQ %00000100       ;STROBE=1
STB0:           .EQ $FF-STB1        ;STROBE=0
CBMBIT:         .EQ %00100000       ;CBM-PRINTER-FLAG
MSBBIT:         .EQ %00010000       ;MSB-OPERATION-FLAG
XFLAG:          .EQ %00001000       ;EXCHANGE-FLAG
;******************************
;* INITIALISIERUNG INTERFACE *
;******************************
;
;
START:          LDA #$FF            ;ALLE DATEN AUSGANG
                STA DATADIR
                LDA STBDIR          ;STB AUSGANG
                ORA #STB1
                STA STBDIR
                LDA STBPORT
                ORA #STB1
                STA STBPORT
;
                LDA #<NEWCHK        ;NEUER CHKOUT
                STA CHKVEC
                LDA #>NEWCHK
                STA CHKVEC+1
                LDA #<NEWOUT        ;NEUER CHROUT
                STA OUTVEC
                LDA #>NEWOUT        ;
                STA OUTVEC+1
                LDA #<NMISER        ;NEUER NMIVEC
                STA NMIVEC
                LDA #>NMISER
                STA NMIVEC+1
                LDA #$7F
                STA CIAICR
```

Listing 7/2.3-1 Teil 2

```
                LDA #IMASK          ;INTERRUPT ENABLE
                STA CIAICR
                LDA #0
                STA QUOTFLG
                STA RVSFLG
                RTS
;************************
;* NMI-INTERRUPT-HANDLER *
;************************
;
NMISER:         PHA                 ;RETTEN DER CPU-REGISTER
                TXA
                PHA
                TYA
                PHA
                LDY CIAICR          ;TEST, OB NMI VOM CIA
                BNE NMIEXIT
                JSR $FD02           ;TEST AUF GAME-ROM
                BNE NOGAME
                JMP ($8002)
NOGAME:         JSR $F6BC           ;STOP-TASTE?
                JSR $FFE1
                BNE NMIEXIT
                JSR $FD15
                JSR $FDA3
                JSR $E518
                JSR START           ;INTERFACE INITIALIZER
                JMP ($A002)
NMIEXIT:        JMP $FEBC           ;SPRUNG AN ENDE NMISEQ
;*****************************
;* CENTRONICS-AUSGABEROUTINE *
;*****************************
;
CENOUT:         BIT CIAICR          ;LESEN DER ACK-LTG
                STA PDATA           ;DATEN ANLEGEN
                LDA STBPORT         ;STROBE 0 IMPULS
                AND #STB0
                STA STBPORT         ;
                JSR DELAY
                ORA #STB1           ;STROBE WIEDER AUF 1
                STA STBPORT
                JSR DELAY
WACK:           LDA CIAICR          ;ACK DA?
                AND #%00010000      ;ACK LEITUNG
                BEQ WACK
DELAY:          RTS                 ; EXIT ZUM AUFRUFER
```

Listing 7/2.3-1 Teil 3

```
;*****************************
;* INTERFACE CHROUT-ROUTINE *
;*****************************
;
NEWOUT:      PHA               ;ZEICHEN RETTEN
             LDA CURDEV        ;DRUCKERAUSGABE?
             CMP PRIDEV
             BNE NOPRINT
             PLA
             STA ZEIBUF
             PHA
             TYA
             PHA
             TXA
             PHA
             JSR PRIOUT
             PLA
             TAX
             PLA
             TAY
             PLA
             CLC
             RTS
NOPRINT:     PLA
             JMP $F1CA         ;WEITER IM ROM
;
;*****************************
;* INTERFACE CHKOUT-ROUTINE *
;*****************************
;
NEWCHK:      JSR 62223         ;FILE OPEN?
             BEQ FILOPEN
             JMP 63233         ;"FILE NOT OPEN!!!"
FILOPEN:     JSR 62239         ;TRANSFER FCDATA
             LDA 186           ;FILE DEVICE#
             CMP PRIDEV
             BEQ PRIFILE
             JMP 62043         ;ANDERE CHKOUT BEHANDELN
PRIFILE:     STA CURDEV        ;DEVICE SPEICHERN
             LDA CURSEC
             AND #$0F
             TAX
             LDA FUNTBL,X
             STA FUNCTION
             CLC
             RTS
```

Listing 7/2.3-1 Teil 4

```
;*********************************
;* KONTROLLE DER DRUCKFUNKTIONEN *
;*********************************
;
PRIOUT:     LDA ZEIBUF         ;ZEICHEN HOLEN
            BIT FUNCTION
            BPL NOCMD
            JMP SPECIAL
NOCMD:      BVS DIRECTP
            CMP #$0D           ;CR?
            BEQ PUTPRT
            CMP #$0A           ;LF?
            BEQ PUTPRT
            CMP #$22
            BNE NOQUOT
            LDA QUOTFLG
            EOR #$FF           ;TOGGLE QOUTFLG
            STA QUOTFLG
            JMP CONVERT
NOQUOT:     LDY QUOTFLG        ;IM QOUTE-MODE?
            BNE CONVERT
            CMP #$8D           ;SHIFT-CR?
            BEQ PUTPRT
            CMP #$8A           ;SHIFT-LF?
            BEQ PUTPRT
            CMP #$12
            BNE NORVSON        ;NICHT RVS ON
            LDA #128
            STA RVSFLG
            JMP CONVERT
NORVSON:    CMP #$92
            BNE CONVERT        ;NICHT RVS OFF
            LDA #0
            STA RVSFLG
            BEQ CONVERT
PUTPRT:     LDY #0             ;FLAGS LOESCHEN
            STY QUOTFLG
            STY RVSFLG
DIRECTP:    JMP CENOUT
CONVERT:    LDA ZEIBUF
            LDY QUOTFLG
            BNE NOTEST         ;KEIN CTRL-TEST
            AND #$7F
            CMP #$20           ;KLEINER BLANK?
            BCS NOTEST
            RTS                ;CTRL NICHT DRUCKEN!
```

Listing 7/2.3-1 Teil 5

```
NOTEST:     LDA  ZEIBUF
            CMP  #$FF
            BNE  WANDELN
            LDA  #94
            STA  ZEIBUF
            JMP  DRUCK
WANDELN:    LSR                  ;ZEIGER BERECHNEN
            LSR
            LSR
            LSR
            LSR
            TAY
            CLC                  ;UMRECHNUNG START
            LDA  ZEIBUF
            ADC  TABELLE,Y
            STA  ZEIBUF
DRUCK:      LDA  FUNCTION        ;TEST FUNCTION
            AND  #CBMBIT         ;CBM-PRINTER AKTIV?
            BEQ  NOCBM
            LDA  ZEIBUF
            ORA  RVSFLG
            STA  ZEIBUF
            JMP  CBMPRI
NOCBM:      LDA  ZEIBUF          ;*** ASCII-PRINTER ***
            BMI  CTRLCODE        ;CTRLCODES->
            JSR  STOASC
            LDA  FUNCTION
            AND  #MSBBIT         ;MSB=1?
            BEQ  NOMSB
            LDA  #%10000000
NOMSB:      ORA  ZEIBUF
            JMP  CENOUT
CTRLCODE:   LDA  FUNCTION
            AND  #%00000011      ;WELCHE ART?
            BNE  DOCTRL
            RTS
DOCTRL:     CMP  #1
            BNE  NOSPC
            LDA  #32
            JMP  CENOUT
NOSPC:      CMP  #2
            BNE  DOROM
            LDA  ENL1
            JSR  CENOUT
            LDA  ZEIBUF
            AND  #$7F            ;BIT 7=0
            STA  ZEIBUF
            JSR  STOASC
```

Listing 7/2.3-1 Teil 6

```
                JSR CENOUT
                LDA ENLO
                JMP CENOUT
DOROM:          JMP CBMPRI
;
STOASC:         LDA ZEIBUF          ;SCREEN TO ASCII
                LSR
                LSR
                LSR
                LSR
                LSR
                STA TEMP
                LDA FUNCTION
                AND #XFLAG
                BEQ NOEXCHG
                LDA #%00000100
NOEXCHG:        ORA TEMP
                TAY
                CLC
                LDA ZEIBUF
                ADC XTABLE,Y
                STA ZEIBUF
                RTS
;*****************************
;* SIMULATION COMMODORE-DRUCKER *
;*****************************
;
CBMPRI:         JSR READROM
                LDY #0              ;INIT GMODE
ILOOP:          LDA GMTABLE,Y
                JSR CENOUT
                INY
                CPY #4
                BNE ILOOP
MATOUT:         LDY #7              ;MATRIXAUSGABE
MLOOP1:         LDX #7
MLOOP2:         ROL MATRIX,X
                ROR
                DEX
                BPL MLOOP2
                JSR CENOUT
                DEY
                BPL MLOOP1
                RTS
READROM:        LDA ZEIBUF          ;ROMBASIS ERRECHNEN
                STA ROMPTR
                LDA #0
                STA ROMPTR+1
```

Listing 7/2.3-1 Teil 7

```
             LDY #3              ;BASIS * 8
MULTIP:      ASL ROMPTR
             ROL ROMPTR+1
             DEY
             BNE MULTIP
             LDA FUNCTION
             AND #%00000100      ;WELCHER SATZ
             BEQ LOWSET
             LDA ROMPTR+1
             ORA #8              ;OFFSET AUF SET2
             STA ROMPTR+1
LOWSET:      CLC
             LDA ROMPTR+1
             ADC #$D0
             STA ROMPTR+1        ;ENDE RECHNUNG
             LDY #7
             SEI
             LDA $01
             TAX
             AND #%11111011      ;CHAREN=0
             STA $01
RELOOP:      LDA (ROMPTR),Y
             STA MATRIX,Y
             DEY
             BPL RELOOP
             STX $01
             CLI
             RTS
;****************************
;* HARDCOPY-ROUTINEN FUER *
;* TEXTSCHIRM, HIRES- UND *
;* MULTICOLORGRAPHIKEN     *
;****************************
;
HGRLF:       LDA #$0D            ;ZEILENVORSCHUB
             JSR CENOUT          ;IM HARDCOPY-MODE
             LDA $B8             ;FILE#
             BPL NOLF
             LDA #$0A            ;LF AUSGEBEN
             JSR CENOUT
NOLF:        RTS
;
ESCOUT:      LDA ESCTBL,Y
             BEQ ESCEXIT         ;ENDE BEI NULL
             JSR CENOUT
             INY
             BNE ESCOUT          ;ENDE BEI NULL INDEX
```

Listing 7/2.3-1 Teil 8

```
ESCEXIT:    RTS
;
;
MCINIT:     LDY #12
            BNE ESCOUT
;
COPYLF:     LDY #3          ;**** ACHTUNG
            BNE ESCOUT      ;**** RELATIVSPRUENGE
;
HGRINIT:    LDY #7
            BNE ESCOUT
;
NORMLF:     LDY #0
            BEQ ESCOUT
;
TXTCOPY:    JSR COPYLF      ;ZEILENABSTAND!
            LDA #25         ;25 ZEILEN
            STA ROWCNT
            LDA 648         ;WO IST VIDEO-RAM?
            STA SCRPTR+1
            LDA #0
            STA SCRPTR
TZEILE:     LDA #40         ;40 SPALTEN
            STA COLCNT
            JSR HGRINIT
ZLOOP:      LDY #0
            LDA (SCRPTR),Y
            STA ZEIBUF
            JSR READROM
            JSR MATOUT
            INC SCRPTR
            BNE SKIP1
            INC SCRPTR+1
SKIP1:      DEC COLCNT
            BNE ZLOOP
            JSR HGRLF
            DEC ROWCNT
            BNE TZEILE
            JMP NORMLF
;
SPECIAL:    LDA ZEIBUF      ;CONTROLROUTINE
            BMI MCCALC      ;MULTI-COLOR-HARDCOPY
            CMP #'*
            BEQ TXTCOPY
            CMP #'1
            BCC SPCEXIT
            CMP #'8
            BCC HGRCOPY
```

Listing 7/2.3-1 Teil 9

```
SPCEXIT:        RTS
MCCALC:         AND #%00111111      ;CALC COLORPAGE
                ASL
                ASL
                STA COLPAGE
                AND #%11100000      ;CALC MAP-AREA
                EOR #%00100000
                STA ROMPTR+1
                LDA #0
                STA ROMPTR
                JMP MCCOPY
;
HGRCOPY:        JSR COPYLF          ;COPYZEILENABSTAND
                LDA #25
                STA ROWCNT
                LDA #0
                STA ROMPTR
                LDA ZEIBUF
                AND #%00000111      ;0..7 BEREICH!
                LSR                 ;UMRECHNUNG ZU PAGE
                ROR
                ROR
                ROR
                STA ROMPTR+1
HZEILE:         LDA #40
                STA COLCNT
                JSR HGRINIT         ;ZEILE EINLEITEN
HLOOP:          LDA $01             ;CPU I/O-PORT
                TAX                 ;RETTEN
                AND #%11111100      ;ALLES RAM!
                LDY #7              ;8-BYTES LESEN!
                SEI                 ;INTERRUPT SPERREN
                STA $01
                JSR RELOOP          ;BITMATRIX LESEN
                JSR MATOUT          ;UND AUSGEBEN
                JSR INCRPTR
                DEC COLCNT
                BNE HLOOP
                JSR HGRLF
                DEC ROWCNT
                BNE HZEILE
                JMP NORMLF
READCLR:        LDA 53281           ;FARBEN HOLEN
                AND #%00001111      ;NUR 4-BIT-REGISTER!
                STA COLFIELD
                LDA CLRPTR+1
                AND #%00000011
                ORA COLPAGE
```

Listing 7/2.3-1 Teil 10

```
                STA CLRPTR+1
                LDY #0
                LDA (CLRPTR),Y
                TAX
                LSR
                LSR
                LSR
                LSR
                STA COLFIELD+1
                TXA
                AND #%00001111
                STA COLFIELD+2
                LDA CLRPTR+1
                AND #%00000011
                TAX
                ORA #$D8            ;COLOR NIBBLES
                STA CLRPTR+1
                LDA (CLRPTR),Y
                AND #%00001111      ;NUR 4-BIT-REGISTER!
                STA COLFIELD+3
                INC CLRPTR
                BNE COLSKIP
                INX
COLSKIP:        STX CLRPTR+1
                RTS
;
MCCOPY:         JSR COPYLF
                LDA #25
                STA ROWCNT
                LDA #0
                STA CLRPTR
                STA CLRPTR+1
MCZEILE:        LDA #40
                STA COLCNT
                JSR MCINIT
MCLOOP:         JSR READCLR
                LDA $01
                TAX
                AND #%11111100
                LDY #7
                SEI
                STA $01
                JSR RELOOP
                LDX #7              ;INTERESSANTER TEIL
CONZEI:         LDA #4
                STA TEMP
CONCLR:         LDA #0
                ASL MATRIX,X
```

Listing 7/2.3-1 Teil 11

```
            ROL
            ASL  MATRIX,X
            ROL
            TAY                     ;ZEIGER AUF FARBEN
            LDA  COLFIELD,Y
            TAY                     ;ZEIGER AUF FARBEN->GRAUSTUFEN
            TXA
            AND  #%00000001
            ASL
            ASL
            ORA  GREYTBL,Y
            TAY                     ;ZEIGER AUF IMAGES
            LDA  GREYIMGS,Y
            LDY  #4
SHFTLOOP:   LSR
            ROL  GREYMAT+8,X
            ROL  GREYMAT,X
            DEY
            BNE  SHFTLOOP
            DEC  TEMP
            BNE  CONCLR
            DEX
            BPL  CONZEI
            LDY  #7
MATMOV1:    LDA  GREYMAT,Y
            STA  MATRIX,Y
            DEY
            BPL  MATMOV1
            JSR  MATOUT
            LDY  #7
MATMOV2:    LDA  GREYMAT+8,Y
            STA  MATRIX,Y
            DEY
            BPL  MATMOV2
            JSR  MATOUT
            CLC
            JSR  INCRPTR
            DEC  COLCNT
            BNE  MCLOOP
            JSR  HGRLF
            DEC  ROWCNT
            BNE  MCZEILE
            JMP  NORMLF
;
INCRPTR:    CLC
            LDA  ROMPTR
            ADC  #8
            STA  ROMPTR
```

Listing 7/2.3-1 Teil 12

```
               LDA ROMPTR+1
               ADC #0
               STA ROMPTR+1
               RTS
;*********************************
;*      DATENSPEICHERBEREICHE      *
;*********************************
;
;STEUERUNG MIT SEKUNDAERADRESSEN
;===============================
;
FUNCTION:      .BY %00100000       ;FUNKTIONSBYTE
;CODIERUNG DES FUNKTIONSBYTES:
;BIT-7:   SONDERFUNKTIONEN (HARDCOPIES)
;BIT-6:   CODE DIREKT ZUM PRINTER
;BIT-5:   COMMODORE-DRUCKER SIMULIEREN
;BIT-4:   MSB-BIT BEI ZEICHENAUSGABE
;BIT-3:   VERTAUSCHEN GROSS-KLEINSCHRIFT
;BIT-2:   CBM-ZEICHENSATZAUSWAHL
;BIT1,0:  BEHANDLUNG VON CTRL-ZEICHEN:
;         00 - UNTERDRUECKEN
;         01 - ALS SPACE DRUCKEN
;         10 - IN BREITSCHRIFT AUSGEBEN
;         11 - AUS CHAR-ROM LESEN
;
;TABELLE MIT FUNKTIONSBYTES
;FUER SEKUNDAERADRESEN 0..15
;===========================
;
FUNTBL:        .BY %01000000       ;SEC# 0
               .BY %00001000       ;SEC# 1
               .BY %00000000       ;SEC# 2
               .BY %00001010       ;SEC# 3
               .BY %00000010       ;SEC# 4
               .BY %00001001       ;SEC# 5
               .BY %00000001       ;SEC# 6
               .BY %00000011       ;SEC# 7
               .BY %00011000       ;SEC# 8
               .BY %00010000       ;SEC# 9
               .BY %00011010       ;SEC# 10
               .BY %00010010       ;SEC# 11
               .BY %00010000       ;SEC# 12
               .BY %00100000       ;SEC# 13
               .BY %00100100       ;SEC# 14
               .BY %10000000       ;SEC# 15
;
```

Listing 7/2.3-1 Teil 13

```
;CENTRONICS-INTERFACE-GERAETENUMMER
;-----------------------------------------------
;
PRIDEV:        .BY 4                    ;PRINTERDEVICE#
;
;ZWISCHENSPEICHER FUER ZEICHEN
;-----------------------------------------
;
ZEIBUF:        .DB 1                    ;ZEICHENBUFFER
;
;1-K-BLOCK# DES MULTICOLOR-FARB-RAMS
;-----------------------------------------------
;
COLPAGE:       .DB 1
;
;FELD FUER MERKEN DER 4 MC-FARBEN
;--------------------------------------------
;
COLFIELD:      .DB 4
;
;UMSETZTABELLE FARBE->GRAUSTUFE
;------------------------------------------
;
GREYTBL:       .BY 3                    ;SCHWARZ
               .BY 0                    ;WEISS
               .BY 3                    ;ROT
               .BY 1                    ;CYAN
               .BY 2                    ;PURPUR
               .BY 2                    ;GRUEN
               .BY 3                    ;BLAU
               .BY 0                    ;GELB
               .BY 1                    ;ORANGE
               .BY 2                    ;BRAUN
               .BY 1                    ;HELLROT
               .BY 2                    ;GRAU 1
               .BY 1                    ;GRAU 2
               .BY 1                    ;HELLGRUEN
               .BY 2                    ;HELLBLAU
               .BY 0                    ;GRAU 3
;
;BITMUSTER DER GRAUSTUFEN 0..3
;(+ VERSCHOBENE WERTE)
;------------------------------------------
;
GREYIMGS:      .BY %00000000
               .BY %01010101
               .BY %11001100
               .BY %11111111
```

Listing 7/2.3-1 Teil 14

```
                .BY %00000000
                .BY %10101010
                .BY %00110011
                .BY %11111111
;
;FELD ZUM AUFBEREITEN DER GRAUSTUFEN
;------------------------------------------------
;
GREYMAT:        .DB 16
;
;FLAGS FUER PRINTERBETRIEB
;------------------------------
;
QUOTFLG:        .DB 1                   ;FLAG FUER QUOTE-MODUS
RVSFLG:         .DB 1                   ;FLAG FUER RVS-MODUS
;
;TEMPORAERE DATENBEREICHE
;----------------------------
;
TEMP:           .DB 1
ROWCNT:         .DB 1                   ;ZEILENZAEHLER
COLCNT:         .DB 1                   ;SPALTENZAEHLER
MATRIX:         .DB 8                   ;ZEICHENMATRIX-BUFFER
;
;TABELLE MIT DRUCKERSTEUERZEICHEN
;------------------------------------
;
ENL1:           .BY $0E                 ;BREITSCHRIFT EIN
ENLO:           .BY $14                 ;BREITSCHRIFT AUS
;
;GRAPHIKMODE EINSCHALTEN FUER 8*8
;------------------------------------
;
GMTABLE:        .BY 27,76,80
;
;ESC-CODE-TABELLE:
;SEQUENZENDE MIT 0
;--------------------
;
ESCTBL:         .BY 27,50,0             ;NORM-ZEILENABSTAND
                .BY 27,65,8,0           ;GRAPH-ZEILENABSTAND
                .BY 27,75,64,1,0        ;HGRMODE EIN
                .BY 27,76,128,2,0       ;DOPPELTE HGR
;
;UMRECHNUNGSTABELLEN
;CBM-ASCII -> BILDSCHIRMCODE
;-------------------------------
;
```

Listing 7/2.3-1 Teil 15

```
TABELLE:        .BY 128,0,192,224
                .BY 64,192,128,128
;
;BILDSCHIRMCODE -> PRINTER-ASCII
;--------------------------------
;
XTABLE:         .BY 96,0,0,192
                .BY 64,0,32,192
                .EN
```

Listing 7/2.3-1 Teil 16

7/2.4

Betriebsumschaltung mittels EPROM

Autor: Ernst Schöberl

Im C 64 und der Floppy befindet sich die Betriebssystemsoftware in ROM's vom Typ 2364. ROM heißt **R**ead-**O**nly-**M**emory, einmal einprogrammierte Daten können also nicht mehr gelöscht werden. Möchte man nun das Betriebssystem seinen eigenen Wünschen anpassen, so bieten sich als günstige Lösung EPROM's als Programmspeicher an. EPROM ist die Abkürzung für **E**rasable **P**rogrammable **R**ead **O**nly **M**emory. Diese Bausteien werden mit einer relativ hohen Spannung, normal 25 V, programmiert. Mit UV-Licht können diese Speicher wieder gelöscht werden.

In letzter Zeit sind sehr viele neue Betriebssysteme und Betriebssystemerweiterungen auf den Markt gekommen. Besitzt man zum Beispiel einen C 64 mit der Floppy 1541, so sind die im Betriebssystem vorhandenen Kassettenroutinen völlig unnötig und man kann sie durch schnelle Floppyroutinen ersetzen. Es gibt Betriebssysteme, die den Ladevorgang bis zu siebenmal schneller machen. Als Kassettenrekorderbesitzer könnte man sich die Turbo-Tape-Routinen direkt ins Betriebssystem einbauen. Es gibt noch eine Reihe anderer Möglichkeiten. Eine davon wird auch in diesem Buch vorgestellt. Es ist die Software für das IEEE-Interface, die in das Betriebssystem eingebunden wird.

Es gibt nur ein Problem, das die Verwendung von EPROM's erschwert. Für das von Commodore verwendete ROM 2364 gibt es kein pingleiches EPROM. Die Adapterplatine wurde für die Verwendung von EPROM's des Typs 2764 ausgelegt. Außerdem kann mit einem Schalter zwischen dem Original-ROM und dem EPROM umgeschalten werden.

Die Funktionsweise der Schaltung ist schnell erklärt: Mit dem Schalter wird das Chipselekt-Signal zwischen dem ROM und dem EPROM umgeschaltet. Die beiden Pullup-Widerstände von 3,3 Kiloohm müssen den Chipselekteingang der IC's auf Highpegel halten, wenn sie nicht selektiert sind, denn dieses ist ein low-aktives Signal. Nach dem Umschalten muß ein Reset gegeben werden. Mit dem Chipselektsignal wird im Computer ausgewählt, welches IC gerade aktiv ist. Es wird durch Dekodierung der Adressensignale gewonnen. Wenn sich die Adresse im Bereich von $E000 bis $FFFF befindet, so wird nur die Chipselektleitung zum Betriebssystem-ROM low, alle anderen bleiben auf Highpegel. Im C 64 übernimmt die Dekodierung ein sogenannter PLA-Chip (Programmable-Logic-Array). Dieses IC ersetzt eine

Vielzahl von herkömmlichen Gatterbausteinen. Alle Speicherbausteine sind mit diesem IC durch eine Chipselektleitung verbunden und werden von ihm entsprechend dem von der CPU angesprochenen Adressenbereich selektiert.

Nun noch ein paar Hinweise zum Aufbau der Platine, der sicher nicht ganz so einfach ist. Nachdem alle Löcher gebohrt sind, beginnt man am besten mit dem Einlöten des 24- und 28-poligen Sockels auf der Bauteilseite. Es folgen die beiden Widerstände. Die Steckleiste wird in zwei Teile von je 12 Stiften Länge geteilt. Es gibt auch spezielle „IC-Sockel", die nach oben und unten Stifte haben. Diese Ausführung ist zwar besser, aber auch teurer. Man sollte darauf achten, daß man die Steckleisten ziemlich senkrecht einlötet, damit sie nachher gut in den Sockel passen. Bei manchen Computern kann es vorkommen, daß das Betriebssystem-ROM nicht gesockelt ist. Wer nicht das entsprechende Gerät zum Auslöten (Saugpumpe, Entlötlitze . . .) des ROM's besitzt, sollte dies lieber von einem Fachmann durchführen lassen. Zuletzt werden an den Umschalter drei Leitungen angelötet. Beim Einlöten der Drähte auf der Platine ist noch zu beachten, daß der Draht, der zur mittleren Lötfahne am Schalter führt, auch mit dem mittleren der drei Lötaugen verbunden ist.

Sollte die Platine nicht einwandfrei funktionieren, so kann dies folgende Ursachen haben: Bei schlechter Ätzung können die Leiterbahnen durch Haarrisse unterbrochen sein. Es können aber auch Verbindungen zwischen den Leitungen existieren, wenn das Kupfer an den freien Stellen nicht vollständig weggeätzt wurde. Diese Fehler spürt man am besten mit einem Durchgangsprüfer oder Ohmmeter auf. Kalte Lötstellen sind eine weitere Fehlerursache, für den Amateur jedoch schwer zu finden.

Zum Schluß nun noch die Stückliste der benötigten Bauteile:

1	IC-Sockel 24-polig
1	IC-Sockel 28-polig
2	Stiftleisten 12-polig
2	Widerstände 3,3 Kiloohm

Anwendungsbeispiele und Anregungen zum Bau eines eigenen Kernals:

1. Wer ein Diskettenlaufwerk besitzt, hat sich sicher schon oft darüber geärgert, daß man nach der Eingabe des Filenamens noch ,8 oder ,8,1 eingeben mußte. Abhilfe schafft das Ändern der Speicherzelle $E1DA von $01 auf $08. In dieser Adresse steht die Geräteadresse, die bei Fehlen der Angaben genommen wird. Die Sekundäradresse steht in Adresse $E1DC. Hier steht normalerweise $01.

2. Vielleicht waren Sie auch schon mal unzufrieden über die blau/blau Farbwahl der Commodore-Programmierer. So können Sie Ihre eigene Einschaltfarbkombination zusammenstellen:

   Nummer der Hintergrundfarbe: $ECDA
   Nummer der Rahmenfarbe: $ECD9
   Nummer der Cursorfarbe: $E535

Das Platinenlayout zu diesem Beitrag finden Sie in Kapitel 7/2.1 Seite 1 und 2.

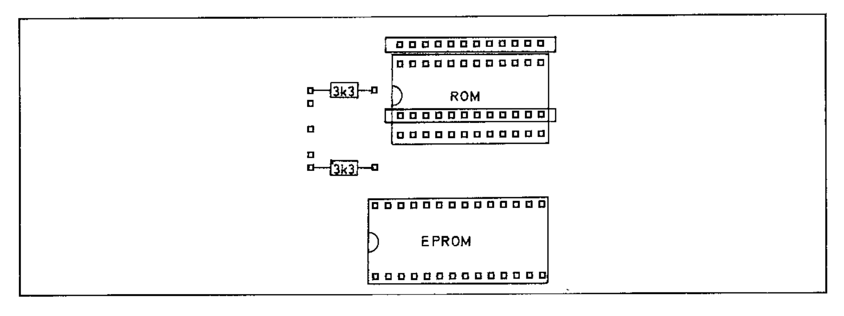


Bild 7/2.4-1 Bestückungsplan (IC-Seite)

7/2.7

Paralleles IEC-Bus-Modul (IEEE-488-Interface)

Autor: Ernst Schöberl

Sicher werden Sie sich fragen, wozu man ein solches Interface überhaupt braucht. Nun, der parallele IEEE-Bus ist die wichtigste Verbindung der Commodore-Computer der 8000er und 4000er Serie mit der Peripherie. Mit diesem Interface können Sie das gesamte Angebot von Commodore-Peripheriegeräten dieser Serien an Ihrem C 64 betreiben. Man kann nun auch die großen Commodore-Floppies wie zum Beispiel die 8250 benutzen und hat so eine Speicherkapazität von einem Megabyte pro Diskette. Zudem bietet der parallele IEEE-Bus gegenüber der seriellen Version eine um mehr als fünffach höhere Übertragungsgeschwindigkeit. Wer sich mit einem Laufwerk zufrieden gibt, kann auch die SFD-1001 anschließen. Außerdem gibt es eine Menge Meßgeräte für den IEEE-Bus, angefangen vom computergesteuerten Digitalvoltmesser bis zur elektronischen Waage.

Der Bau des Interfaces gliedert sich in zwei Abschnitte, dem Aufbau der Hardware und dem Erstellen der Software. Nun zuerst zur Hardware:

Da die Leitungen am Userport nicht ausreichten für eine IEEE-488-Schnittstelle, mußte ein weiterer Schnittstellenbaustein an den C 64 angeschlossen werden. Die Wahl fiel hierbei auf den PIA-Baustein 6821 (PIA = Parallel Interface Adapter). Dieser Baustein ist zum einen recht günstig zu erwerben, zum anderen ist er problemlos an den C 64 anschließbar. Es treten keine Timing-Probleme auf.

Da meist mehrere Geräte am Bus angeschlossen werden, ist noch eine Pufferung der Signale notwendig. Bei den Datenleitungen übernimmt dies der Bustreiber 74LS245. Die Steuerleitungen gelangen über den Inverter/Treiber 7406 auf den Bus. Außerdem schützen die Treiber den 6821 vor Beschädigung.

Aufbau der Platine und Leitungsbelegung

Vor dem Einlöten der Sockel muß man noch beide Steckleistenseiten auf die richtige Breite zufeilen. Das Modul muß sich leicht stecken lassen, darf aber nicht zu locker sitzen, da es sonst zu Fehlverbindungen oder gar Kurzschlüssen kommen kann. Bei Verwendung von nicht durchkontaktierten Platinen, was in der Hobbyanwendung normalerweise der Fall ist, müssen die IC-Sockel auf beiden Seiten angelötet werden. Empfehlenswert ist die Verwendung gedrehter Sockel, die zwar teurer, aber

wesentlich leichter einzulöten sind. Eine andere Möglichkeit ist die Verwendung von Kupferhohlnieten, die in die Bohrungen eingebracht werden. Vor dem Einstecken des Moduls in den Expansionsport des C 64 sollte man noch einmal die richtige Lage der IC's überprüfen. Alle Nasen müssen nach rechts zeigen, wenn das Modul im Port steckt. Bei Fehlern gilt das gleiche, was schon bei der EPROM-Adapterplatine erwähnt wurde. Meist handelt es sich um durch Lötzinnspritzer verbundene Leitungen. Von dem hier verwendeten 6821-Baustein gibt es auch eine stromsparende CMOS-Variante, den 6321. Dieser kann genauso problemlos in dieser Schaltung verwendet werden, nur ist er schwieriger zu erhalten.

Zur Erklärung der Software ist es auch notwendig, daß wir auf die Programmierung des 6821 ein wenig eingehen. Außerdem kann man das IEEE-Interface dann auch als eine Art zweiten Userport anwenden. Der 6821 hat für jeden der beiden Ports drei Register:

1. Das Datenregister enthält die empfangenen oder auszugebenden Daten.
2. Das Datenrichtungsregister gibt an, welche Portbits auf Ein- oder Ausgabe geschaltet sind.
3. Das Kontrollregister verwaltet die Flagleitungen und legt fest, ob das Daten- oder Datenrichtungsregister angesprochen wird.

Im C 64 wurden den Registern folgende Adressen zugeordnet:

$DE00	Daten- bzw. Datenrichtungsregister Port A
$DE01	Kontrollregister Port A
$DE02	Daten- bzw. Datenrichtungsregister Port B
$DE03	Kontrollregister Port B

Mit Bit 2 des Kontrollregisters erfolgt die Unterscheidung zwischen Daten- und Datenrichtungsregister. Ist es gelöscht (0), dann ist das Datenrichtungsregister selektiert. Bit 3 ist für die Flagleitung CA2 bzw. CB2 zuständig. Ist es gesetzt (1), dann liegt an diesem Pin High-Pegel.

Im Datenrichtungsregister ist jeder der acht Ein-/Ausgabeleitungen ein Bit zugeordnet. Ist das Bit gesetzt, so ist die jeweilige Portleitung als Ausgang geschaltet, ist es gelöscht, so fungiert sie als Eingang. Die Bits können völlig unabhänig voneinander geschaltet werden. Es kann zum Beispiel Bit 3 und Bit 5 als Eingang, der Rest als Ausgang geschalten werden. In diesem Falle müssen Sie $D7 in das Datenrichtungsregister eintragen.

Das Datenregister enthält beim Auslesen die Pegelzustände an den als Eingang geschalteten Leitungen. Die Ausgabebits enthalten den hineingeschriebenen Zustand. Beim Schreiben in das Datenregister werden die Ausgabeleitungen auf den ihrem Bit entsprechenden Pegel gesetzt. Die Eingangsleitungen bleiben unbeeinflußt.

Beim IEEE-Interface wurden die Ports folgendermaßen für die Bussignale verwendet:

PA0 bis PA7	DIO 0 bis DIO 7 (Data Input/Output)
CA2	Steuert Datenrichtung des 74LS245
PB0	EOI (End or Identify) Ausgang
PB1	EOI (End or Identify) Eingang
PB2	DAV (Data Valid) Ausgang
PB3	DAV (Data Valid) Eingang
PB4	NRFD (Not Ready For Data) Eingang
PB5	NRFD (Not Ready For Data) Ausgang
PB6	NDAC (Not Data Accepted) Eingang
PB7	NDAC (Not Data Accepted) Ausgang
CB2	ATN (Attention) Ausgang

Da der C 64 als Buscontroller arbeitet, ist nur ein Ausgang für das ATN-Signal notwendig. Mit diesen Informationen wird es Ihnen sicher nicht schwerfallen, das kurze Testprogramm zum IEEE-Bus-Modul zu verstehen. Die eingegebenen Zahlen werden auf den acht Datenleitungen des IEEE-Buses als Binärzahlen ausgegeben. Gibt man 1 ein, so ist nur die Leitung DIO 0 auf Highpegel, DIO 1 bis 7 sind low. Bei 8 ist nur die Leitung DIO 3 high usw. Mit einem Voltmeter kann man die Pegel an den Leitungen überprüfen.

Will man IEEE-Geräte an den C 64 anschließen, so benötigt man noch ein Adapterkabel. Dies ist das gleiche, wie es für die Commodore-Computer der 4000er und 8000er Serie angeboten wird. Man muß dazu einfach von einem TRW-Userportstecker auf einen 24-poligen Amphenolstecker übergehen. Das Verdrahtungsschema für ein Verbindungskabel sieht folgendermaßen aus:

TRW-Stecker	**Signal**	**Amphenolstecker**
Pin 1	DIO 0	Pin 1
Pin 2	DIO 1	Pin 2
Pin 3	DIO 2	Pin 3
Pin 4	DIO 3	Pin 4
Pin 5	EOI	Pin 5
Pin 6	DAV	Pin 6
Pin 7	NRFD	Pin 7
Pin 8	NDAC	Pin 8
Pin 9	IFC	Pin 9
Pin 10	nc	Pin 10
Pin 11	ATN	Pin 11

TRW-Stecker	Signal	Amphenolstecker
Pin 12	GND	Pin 12
Pin A	DIO 4	Pin 13
Pin B	DIO 5	Pin 14
Pin C	DIO 6	Pin 15
Pin D	DIO 7	Pin 16
Pin E-N	GND	Pin 17-24

Die vielen GND-Anschlüsse sind auf der Unterseite der Platine an der IEEE-Steckleiste gut zu erkennen, da die Kontaktflächen untereinander verbunden sind. Mit dieser Orientierungshilfe kann man leicht die anderen Signalleitungen auffinden und mit dem Meßgerät kontrollieren. Die IFC(Interface clear)-Leitung ist identisch mit der Resetleitung im Computer. Alle Signale auf dem IEEE-Bus sind lowaktiv, das heißt, sie sind gültig, wenn sie Low-Pegel haben. Will man z.B. EOI signalisieren, so muß man die Leitung auf Low-Pegel zurücksetzen. Um die Programmierung zu vereinfachen, wurde darum für die Signalleitungen ein invertierender Treiberbaustein verwendet. Die Datenleitungen mußten per Software invertiert werden. Dies geschieht, wie aus dem Assemblerlisting ersichtlich ist, mit dem Befehl EOR $FF.

Benötigte Software

Um Platz für die IEEE-Routinen zu schaffen, wurden aus dem Originalbetriebssystem die Kassettenroutinen entfernt. Da die Software aber nicht den gesamten, freiwerdenden Raum benötigt, wurde noch die Möglichkeit geschaffen, ein eigenes Programm per Tastendruck zu starten. Dies geschieht durch Drücken der Tasten CTRL und '+' gleichzeitig. Die eigene Software darf im Speicherbereich von $F72C bis $FA00 stehen und muß mit einem RTS enden. Denkbar wäre z.B. eine Hardcopy-Routine für Ihren Drucker. Bei Drücken der Tastenkombination CTRL und '+' springt das Programm die Adresse $F72C an. Die eigene Routine muß also hier beginnen. Das Assemblerlisting dient zum Verständnis der IEEE-Routinen und ermöglicht eigenes Verbessern des Programmes. Zusätzlich ist noch ein Basiclader abgedruckt, der schneller einzugeben ist und zudem die Software gleich richtig im Betriebssystem einbindet, nachdem er dieses ins RAM kopiert hat. Besser ist es, man brennt sich ein EPROM von dem Betriebssystem, mit welchem man, mit Hilfe der Adapterplatine, das Original-ROM austauscht. Dies hat den Vorteil, daß die Software sofort nach dem Einschalten aktiviert ist. Der Basiclader besitzt zusätzlich eine Checksumabfrage, die angibt, in welchem Block sich ein Data-Fehler eingeschlichen hat.

Zur Bedienung: Nach dem Reset werden alle Geräte bis auf den Drucker mit Geräteadresse 4 über den parallelen IEEE-Bus angesprochen. In Speicherzelle 2 steht, welche Geräteadresse seriell bleibt. Mit POKE 2,8 wird die serielle Floppy (1541)

mit Laufwerknummer 8 angesprochen. Man kann eine serielle und eine parallele Floppy unter Gerätenummer 8 anschließen und über Speicherzelle 2 immer zwischen den beiden Geräten umschalten, was das Kopieren vereinfacht. Man kann auch von einem seriellen Laufwerk mit der Nummer 8 auf ein paralleles mit der Nummer 9 kopieren und umgekehrt. Will man, daß das parallele Gerät in Speicherzelle 2 steht, muß man in der TESTSER-Routine im Assembler die beiden Bit-Befehle vertauschen.

Noch ein Tip zum Schluß: Stecken Sie das Modul immer nur bei ausgeschaltetem Computer in den Expansionsport, da sonst Bauteile im Computer oder auf der Platine beschädigt werden können. Das gleiche gilt für den Anschluß von Peripheriegeräten, die natürlich auch ausgeschalten sein sollen. Das Platinenlayout zu diesem Beitrag finden Sie im Kapitel 7/2.1 Seite 1 und 2.

Assembler-Listing

```
** Pass 1 **

** Pass 2 **
                ;=================================================================
                ;=                                                               =
                ;=       IEC-Routinen fuer das IEC-BUS-Modul                     =
                ;=         geschrieben von Ernst Schoeberl                       =
                ;=                 im Dezember 1985                              =
                ;=                   Version 2.2                                 =
                ;=                                                               =
                ;=================================================================
                          .os
e1d9                      .ba $e1d9           ;Default auf Floppy umlegen
e1d9 a2 08                ldx #$08            ;Geraeteadresse
e227                      .ba $e227
e227 a2 08                ldx #$08            ;Sekundaeradresse
e5ea                      .ba $e5ea
e5ea 4c ca fb             jmp eigenrout       ;ctrl '+' --> $f72c
eca0                      .ba $eca0
eca0 82                   .by $82
                ;------------------------------------------------------------------
                ;-  Serielle IEC-Routinen umleiten auf parallele Ausgabe  -
                ;------------------------------------------------------------------
ed0e                      .ba $ed0e
ed0e 4c 40 fa             jmp listtalk
edbb                      .ba $edbb
edbb 4c 43 fb             jmp secatn          ;Sekundaeradresse senden
edbe 4c 1c fb             jmp atn0            ;ATN-Signal ruecksetzen
edc1 29 f7                and #$f7
edc3 8d 00 dd             sta $dd00
edc6 60                   rts
edc7 85 95                sta $95
edc9 4c 27 fb             jmp sectalk
ede7                      .ba $ede7
ede7 20 ef fa             jsr sendbyte        ;Byte auf IEC-Bus ausgeben
edea 68                   pla
edeb 85 95                sta $95
eded 18                   clc
edee 60                   rts
edef ea                   nop
edf0 4c fa fa             jmp untalk
edf3 ad 00 dd             lda $dd00
edf6 09 08                ora #$08
```

Listing 7/2.7 Assembler (1)

```
edf8 8d 00 dd              sta $dd00
edfb a9 5f                 lda #$5f
edfd 2c                    .by $2c
edfe a9 3f                 lda #$3f
ee00 4c 06 fb              jmp unlisten
ee03 20 be ed              jsr $edbe
ee06 8a                    txa
ee07 a2 0a                 ldx #$0a
ee09 ca          lab1:     dex
ee0a d0 fd                 bne lab1
ee0c aa                    tax
ee0d 20 85 ee              jsr $ee85
ee10 4c 97 ee              jmp $ee97
ee13 4c 51 fb              jmp readbyte          ;Byte von Bus lesen
                 ;--------------------------------------------------------------
                 ;-        Laden und Saven von Cassette auf Disk umleiten      -
                 ;--------------------------------------------------------------
f4ad                       .ba $f4ad             ;Cass > Disk bei Laden
f4ad c9 04                 cmp #$04              ;Geraeteadresse kleiner 4?
f4af b0 07                 bcs $f4b8             ;nein, dann nicht veraendern
f4b1 a9 08                 lda #$08              ;Geraeteadresse 8
f4b3 85 ba                 sta $ba
f4b5 ea                    nop
f4b6 ea                    nop
f4b7 ea                    nop
f5ef                       .ba $f5ef             ;Cass > Disk bei saven
f5ef c9 04                 cmp #$04
f5f1 b0 07                 bcs $f5fa
f5f3 a9 08                 lda #$08
f5f5 85 ba                 sta $ba
f5f7 ea                    nop
f5f8 ea                    nop
f5f9 ea                    nop
                 ;------------------------------------------
                 ;-  p vor Label heisst parallele Routine -
                 ;------------------------------------------
                 r0:       .eq $de00             ;Register des 6821-Bausteins
                 r1:       .eq $de01
                 r2:       .eq $de02
                 r3:       .eq $de03
f617                       .ba $f617
f617 20 0c fc              jsr savadr            ;Startadr. bei save sichern
fa40                       .ba $fa40
fa40 20 15 fc    listtalk: jsr testser           ;Test: parallel oder seriell
fa43 30 06                 bmi pseclist
fa45 20 a4 f0              jsr $f0a4             ;seriell
fa48 4c 11 ed              jmp $ed11
fa4b 48          pseclist: pha                   ;Sekundaeradr. (nach LISTEN)
fa4c 24 94                 bit $94
fa4e 10 0a                 bpl lab3
fa50 38                    sec
fa51 66 a3                 ror $a3
fa53 20 6e fa              jsr psendbyt
fa56 46 94                 lsr $94
fa58 46 a3                 lsr $a3
fa5a 68          lab3:     pla
fa5b 85 95                 sta $95
fa5d a9 5a                 lda #$5a              ; EOI,DAV,NRFD,NDAC loeschen
fa5f 8d 02 de              sta r2
fa62 a9 08                 lda #$08
fa64 2c 02 de    lab4:     bit r2                ;DAV abfragen
```

Listing 7/2.7 Assembler (2)

```
fa67 f0 fb                          beq lab4
fa69 a9 3c                          lda #$3c       ;atn setzen
fa6b 8d 03 de                       sta r3
fa6e a9 5a          psendbyt:       lda #$5a
fa70 24 a3                          bit $a3
fa72 10 02                          bpl l1
fa74 a9 5b                          lda #$5b
fa76 8d 02 de       l1:             sta r2
fa79 ad 02 de                       lda r2
fa7c 29 50                          and #$50       ;NDAC und NRFD abfragen
fa7e c9 50                          cmp #$50
fa80 f0 56                          beq end1       ;wenn Signale nicht gesetzt
fa82 a9 30                          lda #$30       ;r0 auf Datenrichtungsreg.
fa84 8d 01 de                       sta r1         ;Datenr. des LS 245=Ausgabe
fa87 a9 ff                          lda #$ff       ;alle Bits auf Ausgabe
fa89 8d 00 de                       sta r0
fa8c a9 34                          lda #$34       ;beachtet 245
fa8e 8d 01 de                       sta r1
fa91 a5 95                          lda $95        ;Byte holen
fa93 49 ff                          eor #$ff       ;Byte invertieren
fa95 8d 00 de                       sta r0         ; Byte auf Bus ausgeben
fa98 a9 10          l2:             lda #$10       ;NRFD abfragen
fa9a 2c 02 de                       bit r2
fa9d f0 f9                          beq l2
fa9f a9 04                          lda #$04
faa1 0d 02 de                       ora r2         ;DAV ausgeben
faa4 8d 02 de                       sta r2
faa7 a9 ff          l5:             lda #$ff
faa9 8d 07 dd                       sta $dd07      ;Timer starten
faac a9 19                          lda #$19
faae 8d 0f dd                       sta $dd0f      ;Highbyte
fab1 a9 00                          lda #$00
fab3 8d 07 dd                       sta $dd07      ;Lowbyte des Zaehlers
fab6 ad 07 dd       l3:             lda $dd07
fab9 f0 11                          beq l4
fabb 2c 02 de                       bit r2
fabe 50 f6                          bvc l3         ; wartet auf NDAC
fac0 a9 5a          l6:             lda #$5a
fac2 8d 02 de                       sta r2
fac5 a9 ff                          lda #$ff
fac7 8d 00 de                       sta r0
faca 18                             clc
facb 60                             rts
facc 2c 85 02       l4:             bit $0285      ;time out
facf 10 0a                          bpl end2
fad1 20 e1 ff                       jsr $ffe1
fad4 d0 d1                          bne l5
fad6 f0 03                          beq end2
fad8 a9 80          end1:           lda #$80
fada 2c                             .by $2c
fadb a9 03          end2:           lda #$03
fadd 20 1c fe                       jsr $fe1c
fae0 58                             cli
fae1 18                             clc
fae2 90 dc                          bcc l6
fae4 85 95          l10:            sta $95
fae6 20 6e fa                       jsr psendbyt
fae9 a9 34          patn0:          lda #$34       ;ATN ruecksetzen
faeb 8d 03 de                       sta r3
faee 60                             rts
faef 20 15 fc       sendbyte:       jsr testser    ;Byte auf IEC-Bus senden
```

Listing 7/2.7 Assembler (3)

```
faf2 30 03                 bmi  19          ;parallele Ausgabe
faf4 4c 40 ed              jmp  $ed40       ;serielle Ausgabe
faf7 4c 6e fa   19:        jmp  psendbyt
fafa 20 15 fc   untalk:    jsr  testser
fafd 30 12                 bmi  puntalk
faff 78                    sei
fb00 20 8e ee              jsr  $ee8e
fb03 4c f3 ed              jmp  $edf3
fb06 20 15 fc   unlisten:  jsr  testser
fb09 30 09                 bmi  punlist
fb0b 20 11 ed              jsr  $ed11
fb0e 4c 03 ee              jmp  $ee03
fb11 a9 5f      puntalk:   lda  #$5f
fb13 2c                    .by  $2c
fb14 a9 3f      punlist:   lda  #$3f
fb16 20 4b fa              jsr  pseclist
fb19 4c e9 fa   18:        jmp  patn0
fb1c 20 15 fc   atn0:      jsr  testser
fb1f 30 f8                 bmi  18
fb21 ad 00 dd              lda  $dd00
fb24 4c c1 ed              jmp  $edc1
fb27 20 15 fc   sectalk:   jsr  testser
fb2a 30 06                 bmi  w1
fb2c 20 36 ed              jsr  $ed36
fb2f 4c cc ed              jmp  $edcc
fb32 85 95      w1:        sta  $95
fb34 20 6e fa              jsr  psendbyt
fb37 a9 fa                 lda  #$fa        ;NDAC und NRFD setzen
fb39 8d 02 de              sta  r2
fb3c a9 34                 lda  #$34
fb3e 8d 03 de              sta  r3
fb41 18                    clc
fb42 60                    rts
fb43 20 15 fc   secatn:    jsr  testser
fb46 30 06                 bmi  w2
fb48 20 36 ed              jsr  $ed36
fb4b 4c be ed              jmp  $edbe
fb4e 4c e4 fa   w2:        jmp  110
fb51 20 15 fc   readbyte:  jsr  testser
fb54 30 06                 bmi  preadbyt
fb56 78                    sei
fb57 a9 00                 lda  #$00
fb59 4c 16 ee              jmp  $ee16
fb5c a9 30      preadbyt:  lda  #$30        ;Byte von IEC-Bus lesen
fb5e 8d 01 de              sta  r1
fb61 a9 00                 lda  #$00        ;alle bits auf eingabe
fb63 8d 00 de              sta  r0
fb66 a9 3c                 lda  #$3c        ;Datenr. des LS 245 (bit3) ?
fb68 8d 01 de              sta  r1
fb6b a9 da                 lda  #$da
fb6d 8d 02 de              sta  r2          ; NDAC setzen
fb70 a9 ff      116:       lda  #$ff
fb72 8d 07 dd              sta  $dd07       ;Timer starten
fb75 a9 19                 lda  #$19
fb77 8d 0f dd              sta  $dd0f
fb7a a9 00                 lda  #$00
fb7c 8d 07 dd              sta  $dd07
fb7f ad 07 dd   111:       lda  $dd07
fb82 f0 32                 beq  112
fb84 a9 08                 lda  #$08
fb86 2c 02 de              bit  r2          ;DAV abwarten
fb89 d0 f4                 bne  111
```

Listing 7/2.7 Assembler (4)

```
fb8b a9 fa      225             lda #$fa         ;NDAC und NRFD setzen
fb8d 8d 02 de   226             sta r2
fb90 ad 02 de   227             lda r2
fb93 4a         228             lsr
fb94 4a         229             lsr              ;EOF abfragen
fb95 b0 05      230             bcs l13
fb97 a9 40      231             lda #$40
fb99 20 1c fe   232             jsr $fe1c
fb9c ad 00 de   233  l13:       lda r0
fb9f 49 ff      234             eor #$ff
fba1 48         235             pha
fba2 a9 7a      236             lda #$7a         ;NRFD setzen
fba4 8d 02 de   237             sta r2
fba7 a9 08      238             lda #$08
fba9 2c 02 de   239  l14:       bit r2           ;DAV abwarten
fbac f0 fb      240             beq l14
fbae a9 fa      241  l17:       lda #$fa         ;NRFD und NDAC setzen
fbb0 8d 02 de   242             sta r2
fbb3 68         243             pla
fbb4 18         244             clc
fbb5 60         245             rts
fbb6 2c 85 02   246  l12:       bit $0285
fbb9 10 05      247             bpl l15
fbbb 20 ed f6   248             jsr $f6ed
fbbe d0 b0      249             bne l16
fbc0 a9 02      250  l15:       lda #$02
fbc2 20 1c fe   251             jsr $fe1c
fbc5 a9 0d      252             lda #$0d
fbc7 48         253             pha
fbc8 d0 e4      254             bne l17
fbca c9 83      255  eigenrout: cmp #$83
fbcc d0 03      256             bne testctrl
fbce 4c ee e5   257             jmp $e5ee
fbd1 c9 82      258  testctrl:  cmp #$82
fbd3 d0 06      259             bne lab30
fbd5 20 2c f7   260             jsr $f72c
fbd8 4c fe e5   261             jmp $e5fe
fbdb 4c fe e5   262  lab30:     jmp $e5fe
fbde 46 94      263  taktaus:   lsr $94          ;schafft definierten Zustand
fbe0 a2 38      264             ldx #$38         ; nach Reset oder Close
fbe2 8e 01 de   265             stx r1
fbe5 a2 30      266             ldx #$30
fbe7 8e 03 de   267             stx r3
fbea a9 a5      268             lda #$a5
fbec 8d 02 de   269             sta r2
fbef a9 ff      270             lda #$ff
fbf1 8d 00 de   271             sta r0
fbf4 a2 34      272             ldx #$34
fbf6 8e 01 de   273             stx r1
fbf9 a2 34      274             ldx #$34
fbfb 8e 03 de   275             stx r3
fbfe 8d 00 de   276             sta r0
fc01 a9 5a      277             lda #$5a
fc03 8d 02 de   278             sta r2
fc06 ad 00 de   279             lda r0
fc09 4c 8e ee   280             jmp $ee8e
fc0c a5 c2      281  savadr:    lda $c2
fc0e 85 ad      282             sta $ad
fc10 a5 c1      283             lda $c1
fc12 85 ac      284             sta $ac
fc14 60         285             rts
fc15 48         286  testser:   pha
```

Listing 7/2.7 Assembler (5)

```
fc16 a5 ba      287              lda $ba
fc18 29 0f      288              and #$0f
fc1a c5 02      289              cmp $02
fc1c d0 05      290     test:    bne nn1
fc1e 68         291     tr2:     pla
fc1f 2c 1d fc   292              bit test+1        ;Setzt Negativ Flag zurueck
fc22 60         293              rts
fc23 68         294     nn1:     pla
fc24 2c 1c fc   295              bit test          ;Setzt Negativ Flag
fc27 60         296              rts
fde7            297              .ba $fde7
fde7 a9 04      298              lda #$04          ;Geraet 4 nach Reset seriell
fde9 85 02      299              sta $02
fdeb a9 40      300              lda #$40
fded 4c f3 fd   301              jmp $fdf3
ff7d            302              .ba $ff7d
ff7d 4c de fb   303              jmp taktaus
                304              .en

Fehler:    0
Zeit:      00:04:52
End-Adr.:  ff80/ff80


Label-Files

atn0       p fb1c  eigenrout  p fbca  end1      p fad8  end2      p fadb
l1         p fa76  l10        p fae4  l11       p fb7f  l12       p fbb6
l13        p fb9c  l14        p fba9  l15       p fbc0  l16       p fb70
l17        p fbae  l2         p fa98  l3        p fab6  l4        p facc
l5         p faa7  l6         p fac0  l8        p fb19  l9        p faf7
lab1       p ee09  lab3       p fa5a  lab30     p fbdb  lab4      p fa64
listtalk   p fa40  nn1        p fc23  patn0     p fae9  preadbyt  p fb5c
pseclist   p fa4b  psendbyt   p fa6e  punlist   p fb14  puntalk   p fb11
r0         f de00  r1         f de01  r2        f de02  r3        f de03
readbyte   p fb51  savadr     p fc0c  secatn    p fb43  sectalk   p fb27
sendbyte   p faef  taktaus    p fbde  test      p fc1c  testctrl  p fbd1
testser    p fc15  tr2        u fc1e  unlisten  p fb06  untalk    p fafa
w1         p fb32  w2         p fb4e
```

Listing 7/2.7 Assembler (6)

Testprogramm

```
100 REM ================================
110 REM =   TESTPROGRAMM FUER DAS      =
120 REM =          IEEE-MODUL          =
130 REM =    VON ERNST SCHOEBERL       =
140 REM =            1986              =
150 REM ================================
155 REM
160 REM REGISTERADRESSEN DES 6821 SETZEN
170 R0=13*4096+15*256:R1=R0+1
180 R2=R0+2:R3=R0+3
200 REM EIN-/AUSGABE DATENRICHTUNG SETZEN
210 POKER1,48:POKER0,255: REM ALLE DATENBITS AUSGABE
220 POKER1,52:REM UNSCHALTEN DATENRICHTUNG AUF DATEN
230 POKER3,48 : POKER2,165 : REM STEUERSIGNALDATENRICHTUNG
250 POKER3,52:POKER2,90
260 REM AUSWAHLMENUE
270 PRINT"♥    -------- IEEE-TESTER  -------"
280 PRINT:PRINT:PRINT
290 PRINT "    1 DATENLEITUNGEN SETZEN"
300 PRINT "    2 SIGNALLEITUNGEN SETZEN/LOESCHEN"
310 PRINT "    3 SIGNALLEITUNGEN ABFRAGEN"
320 PRINT
330 PRINT " BITTE ENTSPRECHENDE TASTE DRUECKEN !"
340 REM TASTATURABFRAGE
350 GETA$ : IF A$="" THEN 350
360 IF A$="1" THEN 400
370 IF A$="2" THEN 500
380 IF A$="3" THEN 800
390 GOTO350
400 PRINT "♥  ---- DATENLEITUNGEN SETZEN ----"
410 PRINT:PRINT "EINGABE EINER ZAHL GROESSER 255 BEENDET"
420 PRINT "DAS PROGRAMM !!!":PRINT
430 INPUT Z :REM EINGABE DER ZAHL
440 IF Z<0 OR Z>255 THEN 270
450 POKER0,Z: REM ZAHL AUF DATENBUS LEGEN
460 GOTO 410
500 PRINT "♥--- SIGNALLEITUNGEN SETZEN/LOESCHEN ----"
510 PRINT:PRINT
530 PRINT"  1  ATN SETZEN"
540 PRINT"  2  ATN LOESCHEN"
550 PRINT"  3  NDAC SETZEN  "
560 PRINT"  4  NDAC LOESCHEN"
570 PRINT"  5  NRFD SETZEN  "
575 PRINT"  6  NRFD LOESCHEN"
580 PRINT"  7  DAV SETZEN   "
590 PRINT"  8  DAV LOESCHEN "
600 PRINT"  9  EOF SETZEN"
610 PRINT"  0  EOF LOESCHEN"
620 PRINT "  E ENDE"
630 GETA$: IF A$="" THEN630
640 IF A$="E" THEN 270
650 IF A$="1" THEN POKER3,60
660 IF A$="2" THEN POKER3,52
670 IF A$="3" THEN POKER2,(PEEK(R2)OR128)
680 IF A$="4" THEN POKER2,(PEEK(R2)AND127)
690 IF A$="5" THEN POKER2,(PEEK(R2)OR32)
700 IF A$="6" THEN POKER2,(PEEK(R2)AND223)
710 IF A$="7" THEN POKER2,(PEEK(R2)OR4)
720 IF A$="8" THEN POKER2,(PEEK(R2)AND251)
730 IF A$="9" THEN POKER2,(PEEK(R2)OR1)
```

Testprogramm (1)

```
740 IF A$="O" THEN POKER2,(PEEK(R2)AND254)
750 GOTO630
800 PRINT "♥--- ZUSTAND DER SIGNALLEITUNGEN ---"
810 PRINT:PRINT:PRINT"BEACHTE, DAS BEIM IEEE-BUS GESETZTE"
820 PRINT "SIGNALE LOW-PEGEL FUEHREN!!!"
825 PRINT
830 P=PEEK(R2)
840 IF (P AND 64)=64THEN PRINT "NDAC GELOESCHT (=HIGH)":GOTO860
850 PRINT "NDAC GESETZT   (=LOW)"
860 IF (P AND 16)=16THEN PRINT "NRFD GELOESCHT (=HIGH)":GOTO880
870 PRINT "NRFD GESETZT   (=LOW)"
880 IF (P AND 8)=8THEN PRINT "DAV GELOESCHT  (=HIGH)":GOTO900
890 PRINT "DAV GESETZT    (=LOW)"
900 IF (P AND 2)=2THEN PRINT "EOF GELOESCHT  (=HIGH)":GOTO920
910 PRINT "EOF GESETZT    (=LOW)"
920 PRINT:PRINT
925 PRINT "SPACE FUER WEITERE ABFRAGE DRUECKEN,"
930 PRINT "JEDE ANDERE TASTE FUER ENDE!!"
940 GETA$:IF A$=""THEN940
950 IF A$=" "THEN800
960 GOTO270
```

Testprogramm (2)

Basic-Lader

```
100 POKE56,96:POKE55,0:CLR
110 B=0
120 PRINT"█EINLESEN DER TRANSFERROUTINE!":PRINT
130 RESTORE:FORI=52992TO53068:READA:POKEI,A:B=B+A:NEXTI
140 IFB<>10615THENPRINT"██DATAFEHLER IN BLOCK 1!":END
150 INPUT "RAM-VERSION ODER EPROM VON $6000 BIS    $8000 (R/E)";K$
160 IFK$="R"THENOF=0:GOTO180
170 POKE53048,96:OF=32768
180 SYS52992:REM START DER SCHIEBEROUTINE
190 REM EINSCHALTFARBEN SETZEN
200 POKE58677-OF,5: REM ZEICHEN GRUEN
210 POKE60633-OF,0: REM RAHMEN SCHWARZ
220 POKE60634-OF,0: REM HINTERGR. SCHWARZ
230 PRINT:PRINT"LESEN DER DATA-ZEILEN":PRINT:PRINT
240 T=1
250 T=T+1:READ A:IFA=0THEN380
260 READ B: REM ANZAHL DER BYTES
270 READ P1: REM PRUEFSUMME
280 P2=0:PRINT"BLOCK ";T;"  ";
290 FORI=A-OFTOA-OF-1+B
300 READD:POKEI,D:P2=P2+D
310 NEXTI
320 IFP2<>P1THEN350
330 PRINT"   OK"
340 GOTO250
350 PRINT"PRUEFSUMME FALSCH: ";P2;" STATT ";P1:PRINT
360 GETA$:IFA$=""THEN360
370 GOTO250
380 PRINT:PRINT"FERTIG!"
390 IFK$="R"THENPOKE1,53:POKE2,4:REM KERNALRAM SELEKTIEREN
400 END
1000 REM *****************
1010 REM **** BLOCK 1 ****
1020 REM *****************
1030 DATA165,1,72,169,55,133,1,169,160,141,26,207,169,160,141,29,207,169,192
1040 DATA141,43,207,160,0,185,0,160,153,0,160,200,208,247,238,26,207,238,29
1050 DATA207,173,26,207,201,192,208,232,201,0,240,18,169,224,141,26,207,169
1060 DATA224,141,29,207,169,0,141,43,207,76,22,207,104,133,1,169,229,141,214
1070 DATA253,96
1080 REM *****************
1090 REM **** BLOCK 2 ****
1100 REM *****************
1110 DATA 57817,2,170
1120 DATA 162,8
1130 REM *****************
1140 REM **** BLOCK 3 ****
1150 REM *****************
1160 DATA 57895,2,170
1170 DATA 162,8
1180 REM *****************
1190 REM **** BLOCK 4 ****
1200 REM *****************
1210 DATA 58858,3,529
1220 DATA76,202,251
1230 REM *****************
1240 REM **** BLOCK 5 ****
1250 REM *****************
1260 DATA 60686,3,390
1270 DATA76,64,250
```

Basic-Lader (1)

```
1280 REM *****************
1290 REM **** BLOCK 6 ****
1300 REM *****************
1310 DATA 60576,1,130
1320 DATA 130
1330 REM *****************
1340 REM **** BLOCK 7 ****
1350 REM *****************
1360 DATA 60859,91,11666
1370 DATA76,67,251,76,28,251,41,247,141,0,221,96,133,149,76,39,251,120,32
1380 DATA160,238,32,190,237,32,133,238,32,169,238,48,251,88,96,36,148,48,5
1390 DATA56,102,148,208,5,72,32,239,250,104,133,149,24,96,234,76,250,250,173
1400 DATA0,221,9,8,141,0,221,169,95,44,169,63,76,6,251,32,190,237,138,162
1410 DATA10,202,208,253,170,32,133,238,76,151,238,76,81,251
1420 REM *****************
1430 REM **** BLOCK 8 ****
1440 REM *****************
1450 DATA 62637,11,1586
1460 DATA201,4,176,7,169,8,133,186,234,234,234
1470 REM *****************
1480 REM **** BLOCK 9 ****
1490 REM *****************
1500 DATA 62959,11,1586
1510 DATA201,4,176,7,169,8,133,186,234,234,234
1520 REM ******************
1530 REM **** BLOCK 10 ****
1540 REM ******************
1550 DATA 62999,3,296
1560 DATA32,12,252
1570 REM ******************
1580 REM **** BLOCK 11 ****
1590 REM ******************
1600 DATA 64064,192,22395
1610 DATA32,21,252,48,6,32,164,240,76,17,237,72,36,148,16,10,56,102,163,32
1620 DATA110,250,70,148,70,163,104,133,149,169,90,141,2,222,169,8,44,2,222
1630 DATA240,251,169,60,141,3,222,169,90,36,163,16,2,169,91,141,2,222,173
1640 DATA2,222,41,80,201,80,240,86,169,48,141,1,222,169,255,141,0,222,169
1650 DATA52,141,1,222,165,149,73,255,141,0,222,169,16,44,2,222,240,249,169
1660 DATA4,13,2,222,141,2,222,169,255,141,7,221,169,25,141,15,221,169,0,141
1670 DATA7,221,173,7,221,240,17,44,2,222,80,246,169,90,141,2,222,169,255,141
1680 DATA0,222,24,96,44,133,2,16,10,32,225,255,208,209,240,3,169,128,44,169
1690 DATA3,32,28,254,88,24,144,220,133,149,32,110,250,169,52,141,3,222,96
1700 DATA32,21,252,48,3,76,64,237,76,110,250,32,21,252,48,18,120
1710 REM ******************
1720 REM **** BLOCK 12 ****
1730 REM ******************
1740 DATA 64256,256,31888
1750 DATA32,142,238,76,243,237,32,21,252,48,9,32,17,237,76,3,238,169,95,44
1760 DATA169,63,32,75,250,76,233,250,32,21,252,48,248,173,0,221,76,193,237
1770 DATA32,21,252,48,6,32,54,237,76,204,237,133,149,32,110,250,169,250,141
1780 DATA2,222,169,52,141,3,222,24,96,32,21,252,48,6,32,54,237,76,190,237
1790 DATA76,228,250,32,21,252,48,6,120,169,0,76,22,238,169,48,141,1,222,169
1800 DATA0,141,0,222,169,60,141,1,222,169,218,141,2,222,169,255,141,7,221
1810 DATA169,25,141,15,221,169,0,141,7,221,173,7,221,240,50,169,8,44,2,222
1820 DATA208,244,169,250,141,2,222,173,2,222,74,74,176,5,169,64,32,28,254
1830 DATA173,0,222,73,255,72,169,122,141,2,222,169,8,44,2,222,240,251,169
1840 DATA250,141,2,222,104,24,96,44,133,2,16,5,32,237,246,208,176,169,2,32
1850 DATA28,254,169,13,72,208,228,201,131,208,3,76,238,229,201,130,208,6,32
1860 DATA44,247,76,254,229,76,254,229,70,148,162,56,142,1,222,162,48,142,3
1870 DATA222,169,165,141,2,222,169,255,141,0,222,162,52,142,1,222,162,52,142
```

Basic-Lader (2)

```
1880 DATA3,222,141,0
1890 REM ******************
1900 REM **** BLOCK 13 ****
1910 REM ******************
1920 DATA 64512,40,5061
1930 DATA222,169,90,141,2,222,173,0,222,76,142,238,165,194,133,173,165,193
1940 DATA133,172,96,72,165,186,41,15,197,2,208,5,104,44,29,252,96,104,44,28
1950 DATA252,96
1960 REM ******************
1970 REM **** BLOCK 14 ****
1980 REM ******************
1990 DATA 64992,19,2159
2000 DATA240,0,169,4,133,2,169,37,141,4,220,169,64,141,5,220,76,110,255
2010 REM ******************
2020 REM **** BLOCK 15 ****
2030 REM ******************
2040 DATA 65405,3,549
2050 DATA76,222,251
2060 DATA 0
2070 REM ****************
2080 REM ****  ENDE  ****
2090 REM ****************
```

Basic-Lader (3)

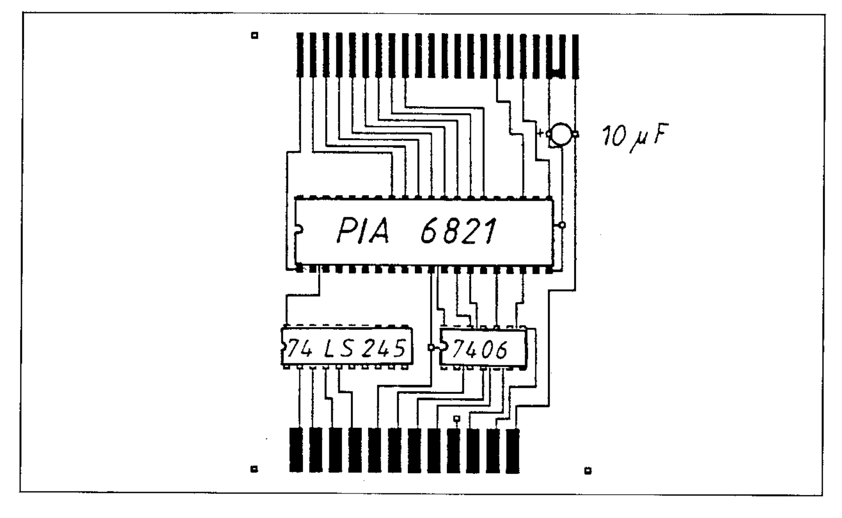


Bild 7/2.7-1 Bestückungsplan

```
1880 DATA3,222,141,0
1890 REM ******************
1900 REM **** BLOCK 13 ****
1910 REM ******************
1920 DATA 64512,40,5061
1930 DATA222,169,90,141,2,222,173,0,222,76,142,238,165,194,133,173,165,193
1940 DATA133,172,96,72,165,186,41,15,197,2,208,5,104,44,29,252,96,104,44,28
1950 DATA252,96
1960 REM ******************
1970 REM **** BLOCK 14 ****
1980 REM ******************
1990 DATA 64992,19,2159
2000 DATA240,0,169,4,133,2,169,37,141,4,220,169,64,141,5,220,76,110,255
2010 REM ******************
2020 REM **** BLOCK 15 ****
2030 REM ******************
2040 DATA 65405,3,549
2050 DATA76,222,251
2060 DATA 0
2070 REM ****************
2080 REM ****  ENDE  ****
2090 REM ****************
```

Basic-Lader (3)

7/2.14

I/O-Schnittstelle für C64

Anschluß des I/O-Bausteins 6821 an den Expansionsport

Diese Schnittstelle ermöglicht dem Computer den Kontakt zur Außenwelt. Sie verbindet externe Geräte (z.B. Bildschirm, Tastatur und Drucker) mit dem System. Um nun mit dem Computer Messungen und Steuerungen durchzuführen, benötigt er einen weiteren Ein-/Ausgabe-Baustein (im Folgenden kurz als „I/O"-Baustein bezeichnet — für Input/Output). Dieser Baustein enthält in der Regel zwei I/O-Kanäle mit einer Breite von acht Bit. Beim C64 kann über den User-Port auf einen Kanal zugegriffen werden. Dieser User-Port reicht aus, um jeweils einen A/D- (= Analog-/Digital-) oder D/A-Wandler anzuschließen. Bei einem Regelvorgang werden aber in vielen Fällen gleichzeitig beide Wandler-Bausteine verwendet. Aus diesem Grunde empfiehlt es sich, einen Schnittstellenbaustein anzuschließen.

Der Baustein 6821

Der I/O-Baustein hat 16 programmierbare Ein-/Ausgänge. Er ist in fast jedem Elektronikversand erhältlich und kostet weniger als 10 DM. Seine Kompatibilität mit den Mikroprozessoren 6502 und 6510 ermöglicht einen problemlosen Anschluß an den C64. Untergebracht ist er in einem 40-Pin-Gehäuse. Intern besteht der Baustein aus zwei I/O-Kanälen (Port A; Port B), von denen jeder drei Register hat. Durch das Datenrichtungsregister (DRRA; DRRB) kann jede der 16 programmierbaren Leitungen als Eingang oder Ausgang festgelegt werden. Das Ausgaberegister setzt den Pegel an den Ausgabeleitungen fest. Der an den Eingabeleitungen anliegende Pegel kann aus dem Ausgaberegister (ARA; ARB) gelesen werden. Mit dem Steuerregister (SRA; SRB) wird zwischen Ausgabe- und Datenrichtungsregister hin- und hergeschaltet.

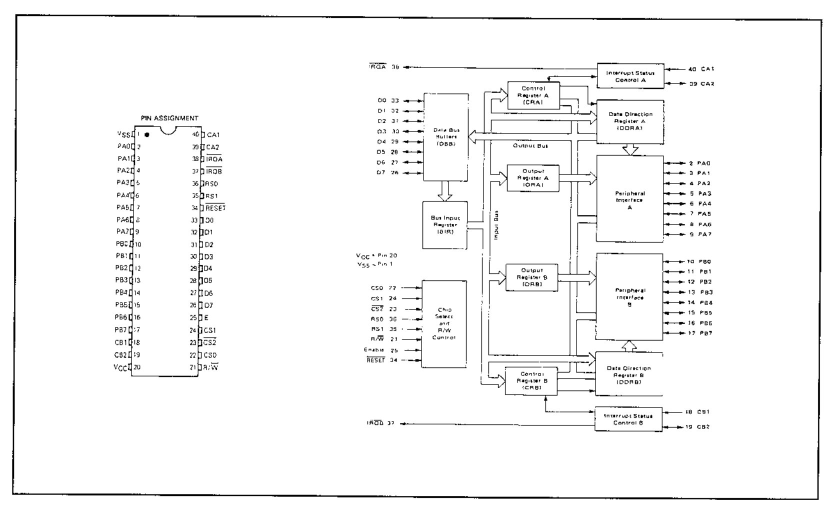


Bild 7/2.14-1: Anschlußbelegung und erweitertes Blockdiagramm des 6821 (Motorola)

Bild 7/2.14-1 zeigt die Anschlußbelegung des Bausteins. Er wird durch Datenleitungen, Adreßleitungen und Steuerleitungen mit dem Computer verbunden. Die Datenleitungen DØ-D7 sind direkt an den Datenbus des Mikroprozessors anzuschließen. Zu den Adreß- und Auswahlleitungen gehören die Anschlüsse RS0, RS1, CS1, CS2 und $\overline{\text{CS3}}$. Zur Aktivierung des Bausteins müssen CS1, CS2 auf +5V und $\overline{\text{CS3}}$ auf GND liegen. RS0 und RS1 werden mit den Adreßleitungen AØ und A1 des Computers verbunden. Die Steuerleitungen $\overline{\text{RES}}$, Ø2 und R/W werden an den entsprechenden gleichnamigen Leitungen des Prozessors angeschlossen. Nach dem Anlegen der Betriebsspannung an Pin 1 (Masse) und Pin 20 (+5V), die dem Computer entnommen wird, ist der I/O-Baustein betriebsbereit.

Wie schon erwähnt, enthält der 6821 sechs interne Register. Man kann jedoch nur vier Register adressieren. So wird durch Umschalten von Bit 2 im Steuerregister das Problem gelöst. Bei dem Baustein kann auf vier Adressen zurückgegriffen werden. Die Adresse ØØ beinhaltet also das DRRA oder das ARA. Auf Ø1 liegt dann das SRA. Die gleiche Adreßbelegung gilt für den Port B (Ø2 = DRRB/ARB; Ø3 = SRB). Um auf das DRRA/B zugreifen zu können, muß man in Bit 2 des SRA/B eine Ø einschreiben. Jetzt kann in das DRRA/B eine achtstellige Binärzahl eingeschrieben werden. Dabei entspricht Ø einer Eingabeleitung und 1 einer Ausgabeleitung. So werden z.B. durch die Binärzahl 1111ØØØ (dezimal = 15) im DRRA PAØ-PA3 als Eingänge und PA4-PA7 als Ausgänge programmiert. Mit dem Einschalten des ARA/B (Bit 2 im SRA/B = 1; dez. = 4) kann der Pegel der als Aus-

gänge geschalteten Leitungen festgelegt werden. Eine 0 im ARA entspricht einem Pegel von 0 V und eine 1 einer Spannung von 4-5 V (TTL-kompatibel). Diese Pegelzustände erkennt der Baustein auch an den Eingangsleitungen. Als Beispiel würde dann ein High-Pegel (2,4-5V) an PA6 in Bit 6 des RA eine 1 schreiben. Die etwas umständliche Programmierung wird später durch Beispiele nochmals veranschaulicht.

Der Expansionsport

Über den Expansionsport kann der Computer bestimmte Erweiterungen aufnehmen. Er ermöglicht den Zugriff zu sämtlichen Adreß-, Daten- und Steuerleitungen des Computers. Außerdem werden dekodierte Leitungen für Speichererweiterung und Schnittstellenerweiterung an die 44polige Buchsenleitung herausgeführt. So reserviert z.B. die dekodierte Leitung $\overline{I/01}$ beim C64 einen Bereich 256 Bytes. Der verwendete Schnittstellenbaustein benötigt einen Bereich von vier Bytes. Es lassen sich also bei vollständiger Dekodierung 16 Bausteine mit einer Auswahlleitung anschließen.

Zum Anschluß des Bausteins wird die Auswahlleitung $\overline{I/01}$ verwendet. Das ergibt die Anfangsadresse 56832. Die Auswahlleitung liegt auf GND. Sie wird mit der CS3-Leitung des Bausteins verbunden. Die restlichen Verbindungsleitungen werden wie oben beschrieben verlegt. Die folgende Tabelle zeigt die vollständige Verbindung des Bausteins mit dem Expansionsport.

Verbindungen

6821	C-64(Exp.)
GND (1)	GND (1)
+5V (20)	+5V (2)
R/W (21)	R/W (5)
CS1 (22)	+5V (2)
$\overline{CS3}$ (23)	$\overline{I/O1}$ (7)
CS2 (24)	+5V (2)
Φ2 (25)	Φ2 (E)
D7 (26)	D7 (14)
D6 (27)	D6 (15)
D5 (28)	D5 (16)
D4 (29)	D4 (17)
D3 (30)	D3 (18)
D2 (31)	D2 (19)
D1 (32)	D1 (20)
DO (33)	DO (21)
$\overline{RES}$ (34)	$\overline{RES}$ (C)
RS1 (35)	A1 (X)
RSO (36)	AO (Y)

Die Anschlußbelegung des Erweiterungsports steht im Handbuch des Computers. Es ist darauf zu achten, daß die Abbildung von der richtigen Seite betrachtet wird. Im C-64-Handbuch ist der Erweiterungsport von vorne abgebildet. Der Datenbus DØ-07 muß von der rechten Seite beginnen, wenn man von oben auf den Computer sieht.

Der Aufbau

Der Expansionsport ist auf eine 44polige Buchsenleiste herausgeführt. Es besteht ein Unterschied zwischen dem Stecker für den VC-20 und den C64: der Stecker für den VC-20 hat ein Raster von 3,96 mm und der für den C64 ein Raster von 2,54 mm. Die Stecker sind nicht bei allen Elektronikversendern erhältlich. Sie müßten aber über ein Computergeschäft zu bekommen sein. Eine andere Lösung wäre ein Selbstbau des Steckers. Man braucht nur eine doppelseitige Platine mit parallellaufenden Bahnen im Raster von 3,96 mm (2,54 mm) zu ätzen.

Ein Versuchsaufbau läßt sich ohne weiteres mit einer Lochrasterplatine anfertigen. Dabei empfiehlt es sich, einen IC-Sockel für den Baustein zu verwenden. Über

ein Flachkabel kann der Stecker mit der Platine verbunden werden. Es ist noch darauf zu achten, daß die Platine vor dem Einschalten angeschlossen wird.

Listing 7/2.14-1 und -2 zeigen ein Testprogramm für den Baustein. Wenn die Programme auf dem C64 laufen, schaltet der Port A (PA0-PA7) zwischen 0 V und 5 V hin und her. Dies läßt sich leicht durch ein Meßgerät feststellen.

```
10 REM *** Ausgabe-Test ***
20 PA=356832
30 POKE PA+1,0
40 POKE PA,255
50 POKE PA+1,4
60 POKE PA,0
70 FORI=OTO2000:NEXT
80 POKE PA,255
90 FORI=OTO2000:NEXT
100 GOTO 60
```

Listing 7/2.14-1

Das zweite Programm testet die Eingänge am Port A. Legt man alle Leitungen (PA0-PA7) auf GND, so gibt der Computer den Zahlenwert Ø aus. Liegt eine Leitung auf +5V und die anderen auf Masse, so ergibt sich der Zahlenwert 2 hoch n (n = PAn). Wurde der Test erfolgreich abgeschlossen, so können bestimmte Anwendungen für den Baustein angeschlossen werden.

```
10 REM *** Eingabe-Test ***
20 PA=56832
30 POKE PA+1,0
40 POKE PA,0
50 POKE PA+1,4
60 PRINT PEEK(PA)
70 GOTO 60
```

Listing 7/2.14-2

Anwendungen des 6821

In Bild 7/2.14-2 werden Schaltungen zum Ansteuern von Leistungsschaltern dargestellt. In der ersten Schaltung wird über einen NPN-Transistor eine LED angesteuert. Die Versorgungsspannung kann bei dieser Schaltung dem Computer entnommen werden. Um größere Leistungen steuern zu können, muß man ein Relais benutzen. Das zweite Schaltbild zeigt die Ansteuerung eines Relais über einen Transistor. Bei der Verwendung von einem Reed-Relais reicht die Versorgungsspannung des Computers aus. Will man aber große Relais (z.B. für Netzspannung) anschließen, muß unbedingt mit einer externen Spannung gearbeitet weden. Es darf auch die Schutzdiode (mind. 1A) beim Schalten von großen Leistungen nicht zu klein sein.

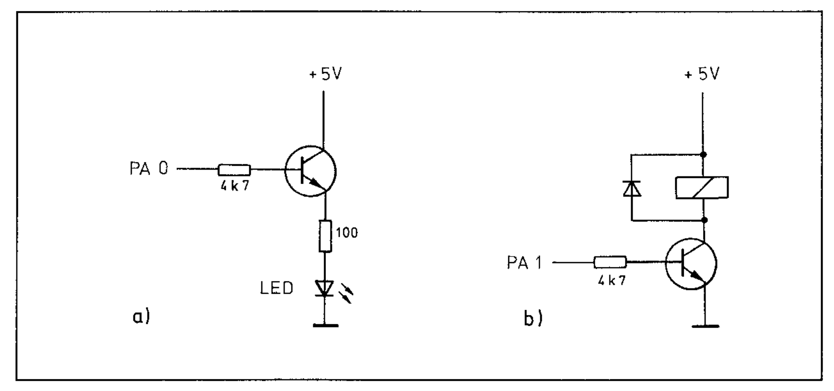


Bild 7/2.14-2: Leistungsschalter mit einem Transistor oder Relais

Anschluß des A/D-Wandlers ADC 0804 an den 6821

Die Möglichkeiten des 6821 sind vielfältig. Eine besonders interessante soll nachfolgend gezeigt werden. Diese Anwendung bezieht sich auf Messungen mit dem Computer. Allgemein ist es möglich, mit dem Schnittstellenbaustein Zeit-, Pegel- und Frequenzmessungen ohne Erweiterungen durchzuführen. Will man aber mit dem Computer Spannungen messen, so ergeben sich zwei Möglichkeiten. Entweder wird die Spannung in eine proportionale Frequenz oder in mehrere parallel anliegende Pegel umgewandelt.

Es gibt verschiedene Techniken der Analog/Digital-Wandlung. Die Methode mit dem direkten Vergleich ist ein etwas umständliches Verfahren und wird deshalb nur selten verwendet. Mehr verbreitet ist das Integrationsverfahren. Es beruht darauf, die Zeit zu messen, die gebraucht wird, einen Kondensator auf die unbekannte Spannung aufzuladen und ihn mit einer bekannten Bezugsspannung wieder zu entladen. Das Zeitverhältnis zwischen diesen beiden Vorgängen entspricht dem Verhältnis der bekannten zur unbekannten Spannung. Die dritte Möglichkeit, ein analoges Signal in ein digitales Signal umzuformen, ist das Verfahren durch die schrittweise Annäherung. Diese Methode ist auch unter dem Namen sukzessive Approximation bekannt.

Sukzessive Approximation

Die A/D-Wandlung nach dem „Wägeverfahren" benutzt einen D/A-Wandler, einen Komparator und eine Steuereinheit. Dabei wird die Eingangsspannung mit der Spannung des D/A-Wandlers verglichen. Der Komparator vergleicht die beiden

Spannungen und sendet dann ein log. Signal an die Steuereinheit. Je nach dem, ob die Eingangsspannung größer oder kleiner ist, erhöht oder erniedrigt die Steuereinheit den Spannungswert für den D/A-Wandler.

Die Genauigkeit des Wandlungsverfahrens bestimmt die Anzahl der zu durchlaufenden Durchgänge. Bei einem 4-Bit-A/D-Wandler muß die Spannung viermal geändert werden, bis sie gleich der Eingangsspannung ist.

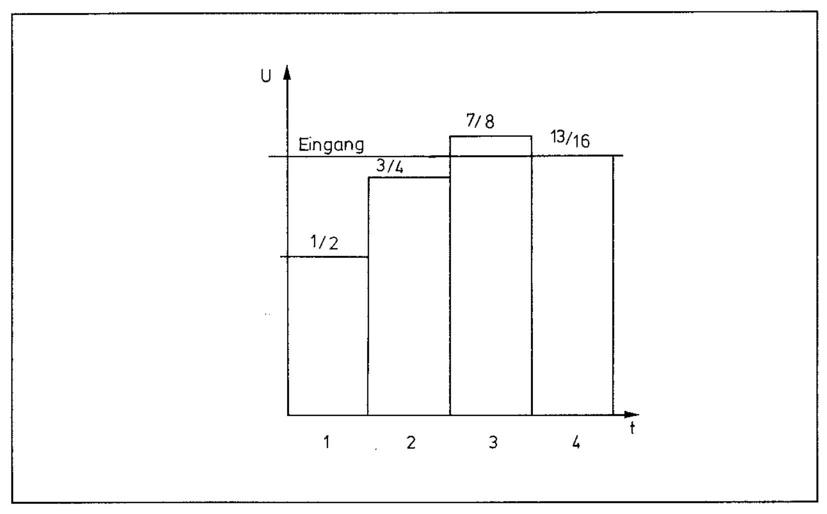


Bild 7/2.14-3: Schrittweises Einstellen der Spannung bei der sukzessiven Approximation

Bild 7/2.14-3 zeigt einen „Ratevorgang" bei schrittweiser Annäherung. Ein Vorteil bei der Umwandlung mit der sukzessiven Approximation liegt in der Geschwindigkeit. Es gibt fertige Bausteine, die Umwandlungszeiten von 10 Mikrosekunden haben. Damit lassen sich Frequenzen im Hörbereich (bis 20 kHz) gut erkennen. Durch den häufigen Gebrauch von A/D-Wandlern in der Computertechnik hat die Industrie komplette integrierte Bausteine entwickelt. Die auf dem Markt erhältlichen ICs haben eine Auflösung von 8–16 Bit. Das entspricht einer Genauigkeit von 1/256–1/65536 des Eingangsspannungsbereiches. Für einfache Messungen kann ein 8-Bit-A/-Wandler ausreichen, um z.B. die Raumtemperatur messen zu können.

Der A/D-Wandler ADC 0804

Der A/D-Wandlerbaustein ADC 0804 von National Semiconductor hat eine Auflösung von 8 Bit und somit eine Genauigkeit von 1/256 der Eingangsspannung.

Seine Kompatibilität zu allen gängigen Computerbussen ermöglichen einen problemlosen Anschluß an den C64. Die Umwandlungszeit des Wandlers beträgt bei Normalbeschaltung (R = 10 k; C = 150 pfF) 100 Mikrosekunden. Der Eingangsspannungsbereich geht von 0 V bis +5 V. Die Versorgungsspannung beträgt ebenfalls 0 V und +5 V. Der Baustein ist bei vielen Elektronik-Versendern erhältlich und kostet ca. 20 DM. Eingebaut ist der Wandler in ein 20-Pin-Gehäuse.

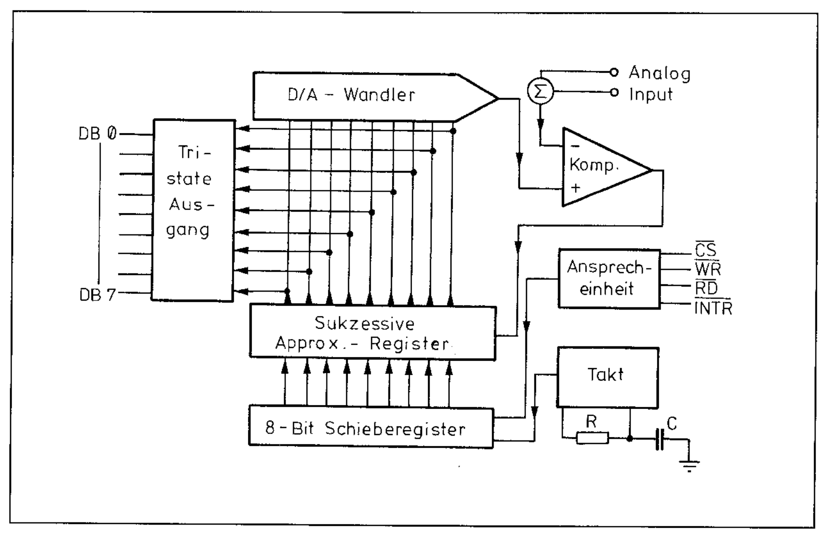


Bild 7/2.14-4: Interner Aufbau des ADC 0804

Der interne Aufbau ist in Bild 7/2.14-4 dargestellt. Es zeigt die Verbindung der einzelnen Komponenten. Will man den Wandler starten, so müssen die Bedingungen der Leitungen an der Ansprecheinheit erfüllt sein. Ist dies der Fall, so geht ein Startsignal von der Ansprecheinheit zum Schieberegister. Damit der Schieberegister auch arbeiten kann, benötigt er einen Takt. Die Taktfrequenz des Taktgenerators kann durch externe Beschaltung von R und C bestimmt werden. Nun ist der Wandler betriebsbereit. Das analoge Eingangssignal wird auf das Nullpotential des Bausteins abgestimmt und kommt dann an den negativen Eingang des Komparators (Vergleicher). Die Ausgangsspannung des D/A-Wandlers geht an den positiven Eingang des Komparators. Die Wandlungslogik wurde weiter oben schon beschrieben. Ist die Ausgangsspannung am D/A-Wandler gleich der Eingangsspannung, so liegt der Wert der Eingangsspannung zwischen sukzessiven Approx.-Register und D/A-Wandler in binärer Form vor. Diese Dualzahl wird dann über das Tri-State Ausgaberegister an die Datenleitungen DBØ-DB7 ausgegeben. Bild 7/2.14-5 zeigt

die Anschlußbelegung des Bausteins ADC 0804. Die externe Beschaltung des Wandlers besteht aus dem Widerstand R = 10 k und dem Kondensator C = 150 pF. Die beiden passiven Bauelemente bestimmen die Taktfrequenz. Der Widerstand darf nicht kleiner als 10 k sein. Der Wert des Kondensators kann noch verkleinert und somit die Taktfrequenz vergrößert werden.

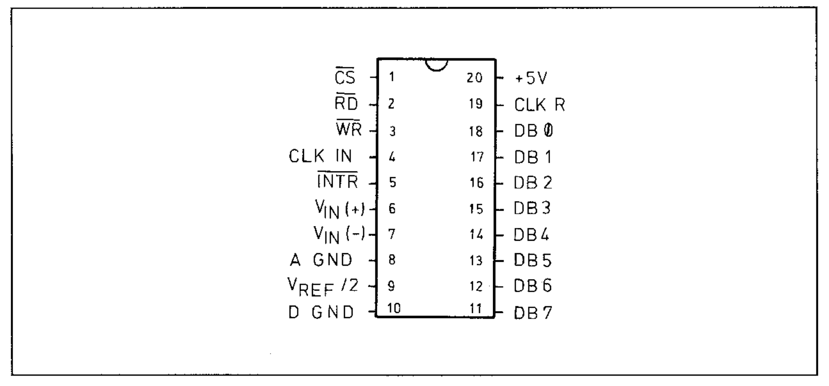


Bild 7/2.14-5: Anschlußbelegung des ADC 0804

Der Aufbau

Da der A/D-Wandler nur in Verbindung mit dem Schnittstellenbaustein arbeitet, empfiehlt es sich, die beiden ICs auf einer Platine aufzubauen. Die Verwendung von IC-Sockeln ist ratsam, da die Bausteine empfindlich und nicht billig sind. Zum Versuchsaufbau kann eine Lochrasterplatine verwendet werden. Bild 7/2.14-6 zeigt die Verbindung des A/D-Wandlers mit dem Schnittstellenbaustein. Der Anschluß des I/O-Bausteins 6821 an den Expansionsport des C64 wurde im ersten Teil der Folge besprochen. Die Betriebsspannung kann bei der Schaltung dem Computer entnommen werden.

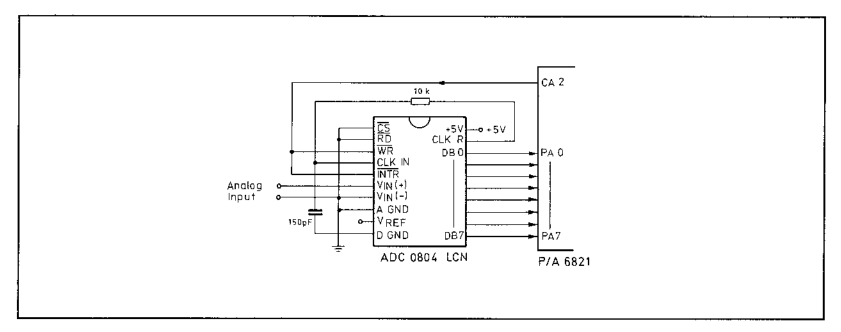


Bild 7/2.14-6: Verbindung des ADC 0804 mit dem P/A 6821

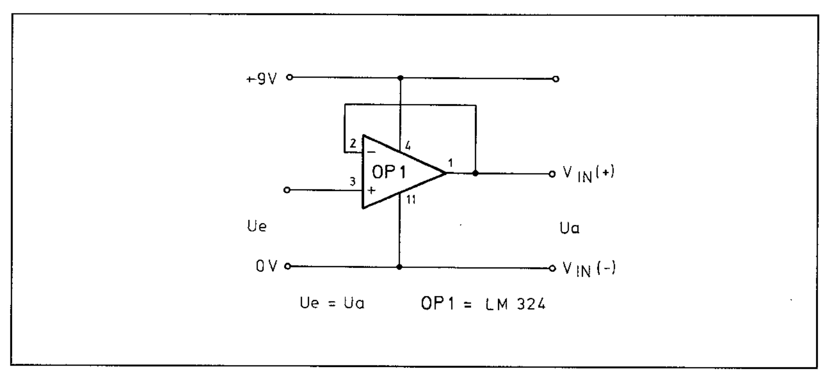


Bild 7/2.14-7: Impedanzwandler

Anwendungen des ADC

Mit dem A/D-Wandler lassen sich direkt Spannungen zwischen 0 V und +5 V messen. Die Schaltung in Bild 7/2.14-7 zeigt einen Impedanzwandler. Der Operationsverstärker arbeitet mit der Verstärkung 1. Wichtig ist die Betriebsspannung von +9V und die Verbindung der Masseleitung des Computers und der Schaltung. Die Schaltung ermöglicht eine belastungsfreie Spannungsmessung. Listing 7/2.14-3 zeigt das entsprechende Programm.

```
10 REM * SPANNUNGS- *
20 REM *   MESSUNG   *
30 REM
40 REM *   WANDLER   *
50 REM *ANSPRECHEN  *
60 PA=56832
70 POKEPA+1,48
80 POKEPA+1,0
90 POKEPA, 0
100 POKEPA+1,4
200 REM *WERT LESEN *
210 PRINT"♥"
220 A=PEEK(PA)
230 B=50/256*A
240 C=INT(B)/10
300 REM * WERT AUSLESEN *
310 PRINT"SQQ" SPANNUNG  ="C"▮▮ VOLT"
320 GOTO220
READY.
```

Listing 7/2.14-3: Spannungsmessung

Mit Sensoren ist es möglich, auch andere Größen zu messen. Auf dem Markt sind Sensoren für Temperatur, Druck, Luftfeuchtigkeit, Magnetfeldstärke und weitere physikalische Größen erhältlich. Die Schaltung in Bild 7/2.14-8 wandelt die Temperaturwerte mit dem Sensor in proportionale Spannungswerte um und paßt die

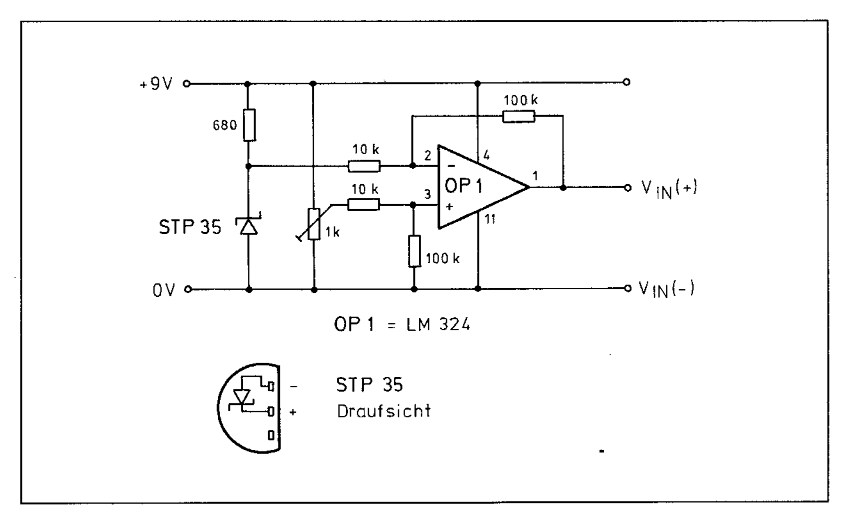


Bild 7/2.14-8: Eingangsschaltung zur Temperaturmessung mit dem Sensor STP 35

umgewandelte Spannung an den Eingangsbereich des A/D-Wandlers an. Der verwendete Temperatursensor STP 35 von Texas Instruments hat eine Empfindlichkeit von 10 mV/K (Kelvin). Die Kalibrierung der Schaltung kann durch Eiswasser oder direkten Temperaturvergleich über den Spindeltrimmer (1 k) erfolgen. Wichtig ist, daß die Eingangsspannung des Wandlers nicht größer als +5 V wird. Darum sollte die Ausgangsspannung der Schaltung erst mit einem gewöhnlichen Meßgerät überprüft werden. Der Operationsverstärker arbeitet mit dem Verstärkungsfaktor 10. So ergibt sich eine Spannungsänderung von 100 mV für 1 Kelvin. Das Programm für die Temperaturmessung ist in Listing 7/2.14-4 dargestellt. In den REM-Zeilen der Programme ist die Funktion des nachfolgenden Programmteils abgedruckt.

```
10 REM * TEMPERATUR- *
20 REM *  MESSUNG    *
30 REM
40 REM *  WANDLER    *
50 REM *ANSPRECHEN   *
60 PA=56832
70 POKEPA+1,48
80 POKEPA+1,0
90 POKEPA, 0
100 POKEPA+1,4
200 REM *WERT LESEN *
210 PRINT"♥"
220 A=PEEK(PA)
230 B=INT((255-A)*.1953)
300 REM * WERT AUSLESEN *
310 PRINT"SQQ" TEMPERATUR ="B"▮▮ GRAD"
320 GOTO 220
READY.
```

Listing 7/2.14-4: Temperaturmessung

Teil 8

Spezielle Einsatzbereiche

8/3 Modelleisenbahnen

Modelleisenbahnen waren sicherlich bis zum Erscheinen des Computers eines der weitverbreiteten Hobbys, wenn nicht das weitverbreitetste. Was liegt näher, das neueste Hobby und eines der traditionsreichsten miteinander zu verbinden. Im Rahmen von Kapitel 8/3 wollen wir uns mit Computersteuerungen von Modelleisenbahnen beschäftigen.

Die Computersteuerung von Modelleisenbahnen bietet den Vorteil, daß man die Zweidimensionalität seines Bildschirms verlassen und echte Abläufe mit seinem Computer steuern kann. Vielleicht kann man auch etwas Familienzusammenführung betreiben, wenn der Vater eine Modelleisenbahn besitzt und der Sohn einen Computer. Vielleicht hat der Vater sogar dann ein Einsehen und beschafft die längst überfällige Floppy.

Zur Zeit sind mehrere elektronische Steuerungssysteme für Modelleisenbahnen auf dem Markt, jedoch nur eines bietet direkt den Anschluß an Computer: Märklin Digital H0. Mit ihm wollen wir uns im Folgenden zuerst beschäftigen.

8/3.1 Märklin Digital H0

Der Marktführer bei den Modellbahnherstellern hat die Vorteile seines Dreileiter-Wechselstromsystems genutzt und dafür eine Digitalsteuerung konzipiert. Die Firma Märklin ist sogar noch einen Schritt weitergegangen und hat zu dieser Digitalsteuerung sofort ein Interface angeboten, das an die verschiedensten Computer angeschlossen werden kann. Vorreiter war auch hier die Gruppe der Commodore Home-Computer.

Nachdem wir die einzelnen Komponenten des Digitalsystems besprochen haben, bieten wir eine kleine Unterprogramm-Bibliothek an und gehen sodann auf spezielle Problemlösungen ein.



Bild 8/3: Der kleine Unterschied: Rechts eine konventionelle Steuerung mit entsprechend vielen Kabeln, links die zwei Kabel von Märklin Digital H0.

8/3.1.1

Die Digital-Geräte

Auch wenn man die Modellbahn direkt — nur über zwei Drähte — mit dem Computer steuern kann, wollen wir in den einzelnen Unterkapiteln die verschiedenen Komponenten des Digitalsystems besprechen. Dadurch läßt sich die Anwendung des Computers viel leichter verstehen.

8/3.1.1.1

Central-Unit

Die Central-Unit ist das Herzstück des Digitalsystems, sozusagen die CPU.



Bild 8/3.1.1.1
Die Zentraleinheit

In ihr laufen alle Fäden zusammen (sprich die Leitungen der Keyboards und Control 80 sowie des Interfaces und der Rückmeldebausteine) und die Information wird über zwei Kabel an die Modellbahnanlage übergeben. Damit werden sowohl Loks als auch Magnetartikel (Weichen, Signale und Fernschalter) gesteuert. Ihren Strom bezieht die Central-Unit von einem Transformer, der nichts anderes ist, als ein Transformator. Gegenüber den üblichen Märklin-Transformatoren weist der Transformer jedoch eine Leistung von 52 VA aus, die speziell auf das Digitalsystem abgestimmt ist.

An die Central-Unit werden die Control 80 und die Keyboards direkt über Steckverbindungen angeschlossen, mittlerweile sind jedoch auch Verbindungskabel erhältlich, sodaß die Geräte auch in einiger Entfernung aufgestellt werden können. Generell werden rechts von der Central-Unit die Control 80 aufgestellt und links die Keyboards. Das Interface wird dabei als Control 80 behandelt.

Neben der Steuerung der Anlage überwacht die Central-Unit auch die angeschlossenen Geräte. Z.B. werden den Control 80 jeweils interne Nummern gegeben. Dadurch ist es möglich, zu verhindern, daß zwei Control 80 auf die gleiche Lok zugreifen. Aber davon später mehr.

8/3.1.1.2

Control 80

Das Control 80 ersetzt den bisherigen Fahrregler und noch ein bißchen mehr.



Bild 8/3.1.1.2
Das Fahrgerät

Gegenüber der analogen Steuerung der konventionellen Fahrregler an den Transformatoren besitzt der Regler des Control 80 diskrete Kontakte. Einer davon dient zum Umschalten der Fahrtrichtung, ein anderer beinhaltet die Nullstellung. Die restlichen vierzehn Kontakte sind für die unterschiedlichen Geschwindigkeitsstufen vorgesehen.

Die Control 80-Geräte werden rechts an die Central-Unit angesteckt, bis zu zehn Stück an der Zahl. Mit ihnen wird nicht nur die Geschwindigkeit der angeschlossenen Loks gesteuert, sondern auch noch einiges, was bisher nicht möglich war: Eine Zusatzfunktion. Diese Zusatzfunktion wird über die beiden Tasten off und function ein- oder ausgeschaltet. Diese Sonderfunktion wird in der Regel das Licht sein — das bei Digital-Loks jeweils nur in Fahrtrichtung leuchtet, aber auch eine Telexkupplung bzw. eine Rauchpatrone bei Dampfloks gelten als Sonderfunktion.

Am Control 80 kann über die Tasten stop/go auch ein Nothalt und ein Neustart bei der Modellbahnanlage erfolgen. In diesem Fall wird die Stromzufuhr zur Anlage abgeschaltet, so daß alle Loks sofort stehen bleiben. Allerdings Weichen kann man dann auch nicht mehr schalten. Mit dem Computer läßt sich dies jedoch sehr einfach umgehen, wie wir im Rahmen von Kapitel 8/3.1.2 noch sehen werden.

Neben den vier genannten Tasten und dem Fahrregler befindet sich noch eine Zifferntastatur von 0 bis 9 an jedem Control 80. Mit diesen Ziffern wird jeweils die aktuelle Loknummer angewählt. Mit dem Fahrregler werden also nicht alle Loks innerhalb eines Stromkreises gesteuert — dies ginge sowieso nicht, da ein konstanter Strom beim Digitalsystem an den Schienen anliegt — sondern jede Lok einzeln.

Dazu besitzt jede Lok einen eigenen Dekoder, den wir im nächsten Kapitel noch kurz besprechen werden. Über das Control 80 rufen Sie also eine der vorhandenen Loks über ihre Nummer auf. Diese Nummer kann Werte von 1 bis 80 (mit Ausnahme von 68) annehmen. Diese und nur diese Lok wird mit dem Fahrregler und der Tastenkombination off/function gesteuert. Wählen sie eine andere Lok, so behält die vorher gewählte Lok ihre Geschwindigkeit und den Einschaltzustand ihrer Zusatzfunktion bei. Dies bedeutet, daß zwar bis zu achtzig Loks einzeln manipuliert werden können, jedoch nur auf eine einzige Lok ein direkter Zugriff besteht.

Haben Sie aber zwei Control 80, so können Sie auch zwei verschiedene Loks direkt beeinflussen, wobei von Hause aus keine Zuordnung stattfindet, welche Lok von welchem Control 80 gesteuert werden kann. Lediglich die Central-Unit verhindert, daß zwei Control 80 auf dieselbe Lok zugreifen.

Nachteil gegenüber einer konventionellen Steuerung ist es, daß jedesmal bei erneutem Einschalten der Anlage jede einzelne Lok aufgerufen werden muß, auch wenn nur ein reiner Blockstreckenbetrieb gefahren wird. Diese Aufgabe kann aber auch der Computer übernehmen.

8/3.1.1.3

Dekoder 80

Der Dekoder 80 ist das Gegenstück zum Control 80. Jede Lok muß einen Dekoder 80 enthalten, um über eine Control 80 angesprochen werden zu können.

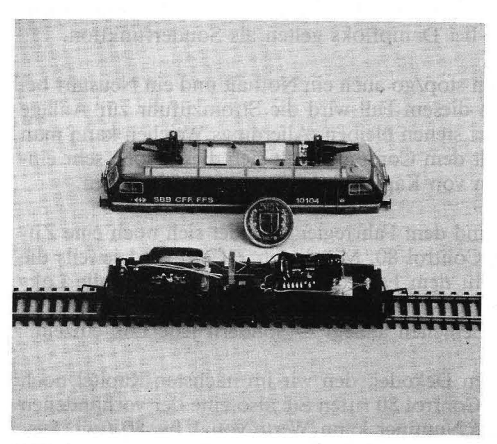


Bild 8/3.1.1.3: Der Dekoder im Entwicklungsstadium

Obwohl an jedem Dekoder 80 ein achtfacher DIL-Schalter angebracht ist, können nicht 255 Loks adressiert werden. Die interne Logik der Dekoder 80 arbeitet mit Tri-State-Technik. Dies bedeutet, daß je zwei Schalter nur drei wirksame Schaltmöglichkeiten besitzen. Fixe Rechner haben sicherlich schnell herausgefunden, daß 3^4 (4 Schaltergruppen mit je 3 Möglichkeiten) 81 ergibt, was die Anzahl der Loks also auf 80 begrenzt, wenn man bei 1 zu zählen beginnt und die Nummer 68 ausspart.

Gegenüber anderen Digitalsystemen bieten die DIL-Schalter eine saubere Lösung, einer Lok eine selbstgewählte Nummer zu geben. Löten von Drahtbrücken oder sogar eine Programmierung der Dekoder ist also nicht erforderlich. Meistens sind die Gehäuse auch mit einer einzigen Schraube am Träger befestigt, so daß hier die wenigsten Probleme anzutreffen sind.

Im derzeitigen Sortiment von Märklin gibt es einige Digital-Loks, u.a. den IC/E. Näheres entnehmen Sie bitte den einschlägigen Prospekten.

Aber auch bereits vorhandene Loks mit konventioneller Steuerung können umgerüstet werden. Bei Drucklegung dieses Werkes gilt dies zwar noch nicht für alle Loks, aber für einen großen Teil. In einem weiteren Schritt werden die Dekoder nochmals verkleinert (im Entwicklungsstadium waren es noch ganze Platinen, siehe Bild Seite 6), sodaß ab Sommer alle Loks auf das Digitalsystem umgerüstet werden können.

8/3.1.1.4

Keyboard

Das Keyboard ersetzt beim Digitalsystem die Weichenstellpulte. Es kann bis zu 16 Magnetartikel schalten.



Bild 8/3.1.1.4
Stellpult

Neben den 32 Tasten zum Schalten der Magnetartikel (für jeden eine rote und eine grüne Taste) befinden sich über der roten Taste noch eine rote Leuchtanzeige, die anzeigt, wenn zuletzt die betreffende rote Taste gedrückt wurde. Dies ist eine einfache Möglichkeit für Rückmeldung der Weichenstellung, sofern man nicht selbst Hand angelegt hat.

Die Keyboards haben auf der Rückseite einen vierpoligen DIL-Schalter, mit dem ihnen Nummern von 0 bis 15 zugewiesen werden können. Daraus resultiert, daß bis zu 255 Magnetartikel an ein Digitalsystem anschließbar sind. Mit einigen einfachen Tricks schafft man auch mehr, wenn das jemand zu wenig ist.

Die Keyboards werden rechts an die Central-Unit direkt angesteckt. Zur Vermeidung von Verwechslungen sind die Keyboards und die Control 80 spiegelbildlich mit Steckern und Buchsen versehen.

Die Signale der Keyboards werden über die Central-Unit direkt an die Dekoder 80 übermittelt.

8/3.1.1.5

Dekoder k83

Auch die Weichen, Signale und andere Magnetartikel müssen in den digitalen Stromkreis einbezogen werden, wenn über die beiden einzigen benötigten Leitungen einzelne Weichen angesprochen werden sollen. Dazu gibt es Dekoder, die die Informationen für vier Magnetartikel aus den digitalen Steuerimpulsen herausfiltern.

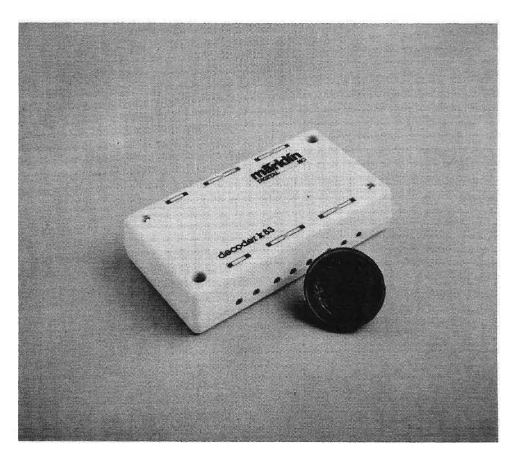


Bild 8/3.1.1.5

Die Dekoder k83 bilden das Gegenstück beim Keyboard gegenüber dem Dekoder 80 zum Control 80. Auch die Dekoder k83 enthalten wieder einen eigenen DIL-Schalter mit 255 Schaltstellungen. Da aufgrund der Anzahl von Keyboards jedoch nur 256 Magnetartikel möglich sind und je Dekoder k83 vier Stück geschaltet werden können, sind nur 64 Schaltstellungen nötig.

Sofern die zwei Kabel von der CPU zur Anlage an der Schiene angeschlossen sind, können die Stromversorgungsanschlüsse der Dekoder k83 ebenfalls irgendwo an den Schienen angeklemmt werden. Wir hatten bereits erwähnt, daß die gesamte digitale Spannung und auch die Steuerinformation ständig an den Schienen anliegt. Die Dekoder k83 können also dort immer exakt vier Magnetartikel, die im entsprechenden Bereich liegen, schalten. Trotzdem vermindern gerade die Dekoder k83 den sonst enormen Kabelaufwand, da nicht mehr für jede Weiche zwei Kabel zum zentralen Stellpult gezogen werden müssen (Ringleitung für den Lichtanschluß vorausgesetzt — sonst drei Kabel).

Obwohl alle Magnetartikel direkt mit einem Dekoder k83 angesteuert werden können, muß auf diese Tatsache nicht immer zurückgegriffen werden können. Im weiteren Verlauf werden wir auf einige Sparmaßnahmen eingehen, um die Kosten des Digitalsystems etwas zu dämpfen.

Teil 9

Kurztests interessanter Standardsoftware

9/1

Kommerzielle Programme

Durch die wachsende Leistungsfähigkeit von Home-Computern werden diese immer öfter im kommerziellen Bereich eingesetzt. Andererseits halten aber auch die kommerziellen Programme Einzug in den häuslichen Bereich, hier sind vor allem die Textverarbeitung und die Dateiverwaltung zu nennen. Neben diesen beiden Bereichen wollen wir auch das Thema Tabellenkalkulation aufgreifen.

9/1.1

Textverarbeitung

Am Anfang ihrer Geschichte waren Computer ‚Nur-Rechner'. Dies änderte sich bei den ersten kommerziellen Einsätzen. Prozentrechnung ist im kommerziellen Einsatz nicht selten die komplexeste mathematische Operation.

Aber mit seinem Siegeszug hat sich der Computer auch weiter von seiner ursprünglichen Aufgabe entfernt. Die meistverbreiteste Heimanwendung ist heute – wenn man einigen Meinungsforschungs-Instituten glauben will – die Textverarbeitung.

Wir haben einige für Sie getestet und wollen diese in loser Reihenfolge vorstellen.

9/1.1.1

Wordpro 3+

(Autor: Thomas Roth)

Wordpro 3+ ist ein semiprofessionelles Textverarbeitungsprogramm, das die Möglichkeiten eines Heimcomputers sehr gut ausnutzt.

Eine Menüführung im herkömmlichen Sinne gibt es nicht. Nachdem man das Programm gestartet hat, legt man den Druckertyp und die Speicherverteilung fest. Fortan besteht der Bildschirm nur noch aus einer Kopfzeile, einer Trennlinie und 23 Schreibzeilen. Befehle, die den Ausdruck betreffen, werden in den Text mit hineingeschrieben. Es wird hierfür ein Zeichen reserviert, das dem Programm anzeigt, daß in dieser Zeile ein oder mehrere Befehle für den Drucker stehen. Standardbefehle, wie Randfestlegen oder Seitengröße bestimmen, sind genauso vorhanden, wie ein Blocksatzbefehl, ein Befehl zum rechtsbündigen Schreiben, ein Befehl zum bedingten Seitenumbruch oder ein Zentrierbefehl.

Der Speicher ist in zwei Teilbereiche aufgegliedert, deren Größe am Programmstart festgelegt wird. Der Hauptspeicher, der normalerweise 300 Zeilen zu 40 Zeichen, also 12000 Bytes faßt, ist für das Erstellen der Haupttexte gedacht. Der Nebenspeicher ist besonders zum Einladen des Inhaltsverzeichnisses der Diskette, sowie für die Daten der Serienbriefe geeignet.

Auch die Möglichkeit, Serienbriefe auszudrucken, ist bei Wordpro 3+ vorhanden. Dies ist ein weiteres und je nach Anwendung auch wichtiges Kennzeichen für ein semiprofessionelles Programm. Beim Erstellen von Serienbriefen werden die Daten, z.B. Anreden und Namen, im Nebenspeicher abgelegt. Beim Text im Hauptspeicher wird an diejenigen Stellen, an denen die Daten nacheinander eingesetzt werden sollen, ein bestimmtes Zeichen gesetzt. Beim Druck werden dann die Daten aus dem Nebenspeicher selbständig in den Text eingefügt.

Es ist nicht möglich, den Text formatiert auf dem Bildschirm auszugeben. Dafür gibt es aber vielfältige Möglichkeiten, den Text während der Erstellung zu verändern. Es besteht die Möglichkeit, ganze Zeilen zu löschen, beziehungsweise ganze Zeilen einzufügen. Auch ein Insertmodus ist vorhanden, in dem man in einen geschlossenen Text etwas einfügen kann, ohne umständlich vorher Platz schaffen zu müssen. Das alles sind aber Funktionen, die in besseren Computern sowieso im Direktmodus verfügbar sind, besonders wenn sie für die Textverarbeitung vorgesehen sind (z.B. CMB 8000, C128, PC's allgemein).

Darüber hinaus gibt es aber bei Wordpro 3+ eben noch eine Vielzahl weiterer nützlicher Funktionen. Es können zum Beispiel Textbereiche markiert und danach gelöscht oder an eine andere Stelle im Text transportiert werden. Ebenso einfach können Wörter oder ganze Sätze gelöscht werden. Der Text kann nach einem festgelegten Begriff durchsucht und dieser gegen einen anderen ausgetauscht werden; dies geschieht nicht nur bei dem Begriff, der so gefunden wird, sondern bei allen, die im Text stehen.

Da der nach dem Laden des Programmes noch übrig bleibende Speicherplatz relativ gering ist, scheint es auch sinnvoll, daß man einen Text in mehrere Stufen gliedern und diese Teile dann zusammenhängend ausdrucken kann. Obwohl das Programm die volle Geschwindigkeit des normalerweise verwendeten Diskettenlaufwerks 1541 ausnutzt, dauert das Laden solcher Texte meist relativ lange. Beim Ausdrucken von längeren Texten ist es natürlich sehr praktisch, wenn der Druckereingang mit einem Puffer versehen ist, weil dadurch die volle Geschwindigkeit des Programmes ausgenutzt wird.

Wie es für ein gutes Textverarbeitungsprogramm kennzeichnend ist, kann auch Wordpro 3+ rechnen. Zahlenkolonnen, die im Text stehen, können sehr einfach aufaddiert werden, was manchmal sehr hilfreich sein kann.

Selbstverständlich ist es, daß man mit diesem Programm auch Diskettenbefehle an das Laufwerk geben und den Diskettenstatus abrufen kann. Meldungen, wie der Diskettenstatus, werden in der Befehlszeile am oberen Bildschirmrand ausgedruckt. Dies erleichtert das Arbeiten sehr, denn man bleibt an seiner Position im Text stehen und muß nicht umständlich erst das Teilprogramm wieder verlassen.

Eine große Anzahl von Fehlermeldungen bei Fehlbedienungen erleichtert den Umgang und die Fehleranalyse. Besonders am Anfang weiß man dies zu schätzen, denn dieses Programm hat zwar den Vorteil, daß man damit sehr schnell und effizient arbeiten kann, allerdings benötigt man auch eine gewisse Einarbeitungszeit, bevor man flüssig mit Wordpro 3+ umgehen kann, aber dies ist bei Textverarbeitungsprogrammen üblich.

Wie auch bei nur-professionellen Programmen besteht hier die Möglichkeit, Kopfzeilen oder Fußzeilen festzulegen. Das sind Zeilen, die jeweils an den Anfang, beziehungsweise an das Ende jeder neuen Seite geschrieben werden. Sehr praktisch ist es, daß man in diesen Spezialzeilen auch die beim Ausdruck aktuelle Seitenzahl einbauen kann.

Eine Fähigkeit, die beim C 64 als einfachen Heimcomputer erwartungsgemäß fehlt, ist der Tabulator. Auch diese Fähigkeit wird durch das Programm simuliert. Mit

wenig Aufwand können direkt dort, wo man mit dem Cursor steht, Tabs gesetzt oder auch wieder aufgehoben werden.

Zusammenfassend kann man sagen, daß dieses Programm für einen 40-Zeichen-Computer sehr passend organisiert ist. Schon nach einer kurzen Einarbeitungszeit wird es Ihnen Spaß machen, zusammen mit diesem Programm Texte zu erarbeiten. Wünschenswert wäre allerdings noch, daß auch im Programm druckerspezifische Zeichen gesendet werden können. Sonst läßt das Programm kaum Wünsche offen.

9/1.2

Kalkulation

Neben der Textverarbeitung erobern sich Tabellen-Kalkulationen (engl. Spread-Sheets) immer mehr Freunde. Kalkulationsprogramme arbeiten mit einem vom Anwender selbst festzulegenden Formular. Mit diesem Formular können ähnliche Rechnungen mit komplizierten Strukturen oder Tabellen sehr leicht durch den Rechner bearbeitet und aktualisiert werden.

9/1.2.1

Multiplan

(Autor: Christoph L. Herd / USA)

Name:
Microsoft MULTIPLAN

Programmart:
Tabellenkalkulationsprogramm

Notwendige Hardware:
Commodore 64 oder 128, Diskettenlaufwerk
Ein Drucker wäre wünschenswert, ist aber nicht unbedingt erforderlich.

Lieferbare Formate:
1541-Format für C 64-Version
CP/M-Format für PC 128-Version

Preis:
etwa 200,- DM

Dokumentation:
Sehr umfangreich, mit Einführungsteil, etwa 200 Seiten Lehrteil, großem Stichwortverzeichnis, vielen Anlagen und umfangreichem Inhaltsverzeichnis.

Technische Daten:
MULTIPLAN ist ein Tabellenkalkulationsprogramm mit 63 Spalten und 255 Zeilen, was dem amerikanischen Standard bei Tabellenkalkulationsprogrammen entspricht.

Das Programm hat 61 Arbeitsfunktionen (z.B. Load, Save, Print, etc.) und 40 mathematische Funktionen (wie z.B. sin, cos, sum, etc.), davon sind 8 logische Vergleiche (IF-THEN-ELSE, AND, OR, NOT, TRUE und FALSE, ISERROR und ISNA). Zusätzlich gibt es natürlich auch die bekannten Operatoren “+“, “’“, “*“, „/“, „=“, „<“, „>“, „<=“, „>=“ und „<>“.

Generelle Beurteilung:

Bevor wir das Programm näher beschreiben, möchten wir auf die zwei möglichen Formate eingehen. MULTIPLAN ist sowohl für den Commodore 64 erhältlich, als auch für den C 128. Die C 64-Version läuft auf der 6510-CPU und mit dem 40-Zeichen-Bildschirm, was bei Tabellenkalkulationen leider sehr wenig ist. Besitzer eines C 128 können sich hier freuen, MULTIPLAN ist für den C 128 unter dem CP/M-Format erhältlich, d.h. schnellere Geschwindigkeit und voller 80-Zeichen-Bildschirm. Vom Bedienungsablauf unterscheiden sich die Versionen nicht, die C 64-Version hat eben nur einen kleineren Bildschirm und ist (bedingt durch das langsamere 1541-Floppy-Laufwerk) auch etwas langsamer. In der Rechenleistung gibt es aber keine Abstriche zur 128-CP/M-Version.

MULTIPLAN auf den Commodore-Heimcomputern (C 64 & C 128) ist somit eine vollständige Implementierung, in der zum Vergleich eines MULTIPLAN’s für den IBM-PC nichts fehlt. Wie kann man aber ein derart umfangreiches Programmpaket wie MULTIPLAN auf einem Homecomputer zum Laufen bringen? Die Entwickler haben hier zu einem Trick gegriffen: **Programm-Overlays.**

Bei dieser Methode wird ein Management-Programm in den Computer eingelesen. Wenn nun eine Funktion (z.B. „Save“) gewählt wird, dann lädt dieser Manager den Programmteil ein, welcher zur Abwicklung dieser Funktion notwendig ist. Bei der Groß-DV sagt man hierzu: Das Programm steht in einem virtuellen Speicher.

Diese Programmiertechnik bedingt natürlich, daß die Systemdiskette mit der Funktionsbibliothek sehr oft angesprochen wird. Bei einem Einzellaufwerk behilft man sich dadurch, daß man die Bibliothek auf die Datendisk mitkopiert, dadurch geht zwar auf der Datendisk etwas Platz verloren, man spart sich aber auf der anderen Seite sehr viele Diskettenwechsel. Bei C 64-MULTIPLAn hat die Funktionsbibliothek den Namen „MP, SYS“ und ist eine REL-Datei.

Im generellen ist dieses Programm wahrscheinlich das Beste und Vielfältigste, was pure Leistung betrifft. Außer den vielen normalen und erwarteten Tabellenkalkulationsprogrammfunktionen kann man zahlreichen exotischen Funktionen begegnen. So kann man den Bildschirm in acht Windows aufteilen, Daten von einer

Tabelle in einer anderen Tabelle verwenden, Daten sortieren, Spalten und Zeilen mit Namen benennen und die Tabelle (bei MULTIPLAN heißt sowas „Worksheet“ oder auch „Arbeitsblatt“) als SEQ-Datei auf Disk ausgeben. Diese Datei kann dann von Textverarbeitungsprogrammen weiterverarbeitet werden.

Der Autor verarbeitet solche Dateien sowohl mit PaperClip von Batteries Included, als auch mit Easy-Script oder dem legendären Wordpro 3+/64. Manchmal kann es Probleme geben, wenn z.B. die Textverarbeitung die Großbuchstaben der Arbeitsblatt-Datei nicht schlucken will. Dies kann daran liegen, daß Commodore zwei verschiedene ASCII-Code-Bereiche für Großbuchstaben verwendet, manche Textprogramme aber nur einen Bereich akzeptieren. In solchen Fällen hilft ein einfaches BASIC-Programm, das die entsprechenden Codes in ein für die Textverarbeitung akzeptables Format umsetzt.

Als weitere Stärken von MULTIPLAN ist die variable Spaltenweite und die Fähigkeit zur Iteration zu nennen. Iteration ist das Wiederholen eines bestimmten Rechenvorgangs bis ein gewünschtes Ergebnis vorliegt.

Das Programm arbeitet sehr zuverlässig und es ist schlechthin unmöglich, es versehentlich zum Absturz zu bringen. Benutzerfehler, wie zum Beispiel das Speichern eines Worksheets auf einem Disk-Laufwerk ohne eingelegte Diskette, das Ausdrucken ohne angeschlossenem Drucker oder das Speichern auf schreibgeschützten Disketten, werden abgefangen und führen nicht zum Programm-Crash. Auch ein „Panik-Knopf“ ist zu finden (RUN/STOP-Taste beim C 64). Diese Taste bringt einen immer zum Hauptmenü zurück. Das ist bei weitem jedoch noch nicht alles. Wenn man irgendwo nicht mehr weiter weiß, dann genügt ein einfacher Druck auf „?“, und MULTIPLAN liefert sofort alle nötigen Informationen zu der unklaren Funktion. Diese Hilfstexte werden aus einer anderen MULTIPLAN-Bibliothek geholt („MP.HLP“). Wer mit MULTIPLAN noch wenig Erfahrung hat, sollte sich diese Bibliothek auch auf seine Datendisk kopieren, MULTIPLAN arbeitet auch ohne diese Bibliothek, aber dann ist das Abrufen von Hilfstexten und Erklärungen logischerweise nicht mehr möglich.

Stärken:

Leistung ist das Hauptmerkmal dieses Programms. Wenn etwas mit einem Tabellenkalkulationsprogramm machbar ist, dann kann man es mit MULTIPLAN machen. Angenehm ist, daß man innerhalb des Ausdruckmenüs noch Kommandos an den Drucker senden kann, so daß eine besonders große Tabelle im 132-Zeichen-Mode gedruckt werden kann. Ein weiteres Plus ist die Erhältlichkeit von Trainingsbüchern von Drittherstellern, hervorgerufen durch die Popularität dieses Programms. Das Bedienungshandbuch enthält u.a. einen Anhang in dem beschrie-

ben wird, wie man VISICALC-Kommandos in MULTIPLAN-Kommandos übersetzt. Dieser Anhang ist wichtig für Benutzer, die entweder von VISICALC auf MULTIPLAN umsteigen, oder die VISICALC tagsüber im Büro benutzen und sich Arbeit für ihren Heimcomputer mit nach Hause nehmen wollen. Auch Tabellenbeispiele, die in Anwendungsmagazinen in VISICALC-Format besprochen und abgedruckt sind, sind dadurch übersetzbar.

Schwächen:

Die enorme Leistung dieses Programms wird durch seine Langsamkeit erkauft. Jedes Mal, wenn ein neues Kommando ausgeführt wird, muß der dazu notwendige Programmcode vom Diskettenlaufwerk gelesen werden, was in einer etwa 5 bis 6 Sekunden langen Pause resultiert. Auch beim Laden eines Arbeitsblattes wird alles geladen, jedes eingestellte Format, jedes Window, einfach alles. Dadurch wird auch das Laden und Speichern von Arbeitsblättern sehr langsam.

Das Handbuch enthält so ziemlich alles, was es über MULTIPLAN zu sagen gibt. Leider ist es mit über 400 Seiten zu dick für den Durchschnittsbenutzer, der eine schnelle Übersicht über die Funktionen erwartet.

Die Bildschirmfarben (Rand-, Hintergrund- und Zeichenfarbe) bleiben bestehen und können vom laufenden Programm aus nicht verändert werden. Man darf deshalb nicht vergessen, sich vor dem Start von MULTIPLAN eine angenehme Farbkombination einzustellen.

Bei der C 64-Version wird es auf dem 40-Zeichen breiten Bildschirm naturgemäß etwas eng, und man braucht schon einige Phantasie, um sich vorzustellen, was geschieht, wenn man alle 8 möglichen Windows aufmachen will. Die C 128-Version hat diesen Nachteil glücklicherweise nicht.

Gesamtbeurteilung:

Unserer Meinung nach ist Multiplan das stärkste Tabellenkalkulationsprogramm für die Commodore-Heimcomputer. Die hohe Leistung wird jedoch durch Geschwindigkeitseinbußen bezahlt, aber das sollte bei Heimanwendungen keine so große Rolle spielen. Dieses Programm, sowie das dazugehörige Handbuch ist jedoch nicht für den gelegentlichen Benutzer gemacht, sondern erfordert gründliche Einarbeitung (Einarbeitungszeit: etwa 2 Tage). Wer ein anderes, einfacher zu bedienendes Programm kennt, das seine Wünsche voll befriedigt, der sollte auf MULTIPLAN verzichten. Wer aber ausgefallene Wünsche hat und hohe Leistungsfähigkeit bei seinen Kalkulationsaufgaben verlangt, der wird an MULTIPLAN wohl nicht vorbeikommen.

9/1.3

Dateiverwaltung

Neben der Textverarbeitung und dem Spieleinsatz wird ein Home-Computer am häufigsten für die Dateiverwaltung verwendet. Ob dies nun im kommerziellen Einsatz für Kundenadressen ist, oder im privaten Bereich für die Adressen und Telefonnummern von Bekannten oder noch weiter zur Verwaltung von Dias, Schallplatten, Tonbändern oder Briefmarken, ist den meisten käuflichen Standardpaketen egal. Hier kann der Anwender seinen ganz speziellen Datenaufbau realisieren.

Manche Dateiverwaltungen für Home-Computer nennen sich großspurig Datenbank, obwohl weder die Hardware- noch Softwarevoraussetzungen vorhanden sind. Was eine Dateiverwaltung wirklich leistet, ist vom praktischen Einsatz und dem Anwendungsgebiet abhängig. Alles muß aufeinander abgestimmt sein und der Anwender eines C 64 oder C 128 braucht in den seltensten Fällen eine komplizierte Datenbank.

Ergänzend zu der in diesem Buch vorgestellten umfassenden Dateiverwaltung möchten wir einige Standardpakete für Sie testen und vorstellen.

9/1.3.1

Multidata

(Autor: Thomas Roth)

Multidata ist ein kombiniertes Datenverwaltungs- und Textverarbeitungsprogramm.

Auffällig ist zuerst, daß dieses Programm mehrsprachig in der Benutzerführung ist. Man kann zwischen deutsch, englisch, französisch, italienisch und spanisch

auswählen. Das Programm benötigt nicht zuletzt wegen dieser Mehrsprachigkeit eine ganze Diskettenseite auf der 1541. Dadurch ist - bei nur einem Laufwerk - ein häufiges und lästiges Wenden oder Wechseln der Diskette nicht zu umgehen, weil jedes neue Teilprogramm von der Programmdiskette gelesen werden muß.

Mit Multidata können bis zu 16 Datenmerkmale je Datensatz frei festgelegt werden. Dazu legt man zuerst den Namen des Datenmerkmals (z.B. Nachname) fest und gibt dann die gewünschte Form ein. Als Form stehen ‚alpha' (Buchstaben), ‚numerisch' und ‚datum' zur Auswahl. Das Datenmerkmal kann bis zu 25 Zeichen umfassen, was manchmal jedoch nicht ausreichend ist. Es können auch bestimmte Datenmerkmale als Indexfelder festgelegt werden. Das bedeutet, daß diese getrennt in einer Indexdatei abgespeichert werden und man später einen Datensatz recht schnell anhand eines solchen Indexfeldes suchen kann.

Bemerkenswert ist auch, daß immer, wenn eine bestimmte Datei zur Bearbeitung ausgesucht werden soll, alle Dateien von dieser Diskette auf dem Bildschirm aufgeführt und mit einer Nummer versehen werden. Nun muß man nur noch diese Nummer eingeben und erspart sich die Schreibarbeit für den ganzen Dateinamen. Dies erscheint vielleicht im ersten Augenblick sehr unwichtig, es hat aber besonders den Vorteil, daß man einen Überblick von den auf der Diskette befindlichen Dateien erhält.

Es besteht auch die Möglichkeit, die Datensätze in einer Datei zu sortieren. Als Sortierkriterium können mehrere Felder herangezogen werden, die dann mit unterschiedlichen Prioritäten beim Sortieren versehen werden. Neben dem üblichen aufsteigenden Sortieren ist auch ein abfallendes Sortieren möglich.

Die Funktionstasten sind mit häufigen Funktionen wie ‚zurück zum Hauptmenü', ‚zurück zum letzten Menü' oder ‚Ton ein/aus' belegt. Diese Belegung ist immer in den unteren zwei Bildschirmzeilen angezeigt. Das verkleinert zwar den effektiv nutzbaren Platz auf dem Bildschirm, aber dadurch und durch die gute Menüführung ist für das Programm kaum eine Einarbeitungszeit nötig. Das Eingabesystem im allgemeinen ist benutzerfreundlich und einfach zu handhaben.

Einen eigenen Programmteil gibt es für Liste/Aufstellung erstellen. Hiermit kann man die Daten oder Teile der Daten einer Datei auf dem Bildschirm oder auf dem Drucker ausgeben. Zuerst wird festgelegt, welche Datenmerkmale stellvertretend für jeden Datensatz ausgegeben werden sollen. Dann wird festgelegt, nach welchen Auswahlkriterien die Datensätze ausgesucht werden sollen (Selektieren der Datensätze). Zum Beispiel könnte man hier festlegen, daß nur die Datensätze mit einem Geburtsdatum vor dem 01.01.1950 ausgegeben werden sollen. Beachtlich ist, daß einzelne dieser Kriterien auch mit ‚und' bzw. ‚oder' miteinander verbunden werden können. Danach wird noch gefragt, die Daten

in der Sortierung eines Indexfeldes oder in der Reihenfolge, die von einer Sortierung bestimmt ist, ausgegeben werden sollen. Dieser Programmteil ist sicherlich einer der besten im ganzen Programm.

Es besteht auch die Möglichkeit, eine ganze Diskette oder eine Datei von Multidata aus auf ein anderes Laufwerk zu kopieren. Ein speichergepuffertes Kopieren mit nur einem Laufwerk, ist leider nicht möglich.

Wie schon am Anfang erwähnt, besitzt Multidata auch die Möglichkeit, Texte zu verarbeiten. Hier können Texte erstellt, gespeichert, geändert und ausgedruckt werden. Dieser Programmteil ist aber nur mäßig ausgebildet. Die Texte müssen zeilenweise eingegeben werden und ein freies Bewegen im Text ist dadurch nicht möglich. Insgesamt erscheint dieser ganze Programmteil unbeholfen und umständlich. Jedes durchschnittliche Textverarbeitungsprogramm bietet mehr an Leistungsfähigkeit.

Zusammenfassend ist zu bemerken, daß der Datenverarbeitungsteil recht gut und komfortabel ausgebildet ist, dagegen ist der Textverarbeitungsteil zu einfach. Auch die Rechengeschwindigkeit im Programm läßt zu wünschen übrig. Zusammen mit dem langsamen Diskettenlaufwerk wird es bald langweilig, mit diesem Programm zu arbeiten.

Teil 10

Tabellen und Diagramme

10/3 Maschinensprache-Befehle (Mnemonics)

Im Rahmen der Übersichten dürfen natürlich auch die Maschinensprache-Befehle nicht fehlen. Dabei wollen wir zwei verschiedene Übersichten darstellen, zum einen eine alphabetische Befehlsübersicht, zum anderen eine Übersicht nach Opcodes geordnet.

Alphabetische Befehlsübersicht der Maschinensprache-Befehle

In der folgenden Tabelle haben wir für Sie alle Maschinensprache-Befehle in alphabetischer Reihenfolge zusammengefaßt. Um Ihnen ein möglichst einfaches Arbeiten mit dieser Übersicht zu gestatten, haben wir neben den Befehlskürzel auch die englische Bedeutung (aus der sich meist die Abkürzung ergibt) und die deutsche Bedeutung eingetragen. Außerdem ist die Funktion angegeben, wobei als Variablen die in Kapitel 4/5 angegebenen Werte zugrunde liegen.

Sehr wichtig für eine Übersicht ist natürlich die Behandlung der Flags. Um auch auf die Unterschiede der Beeinflussung eingehen zu können, haben wir auf drei Kürzel zurückgegriffen:

a — aktiv: Das Flag **beeinflußt** die Wirkungsweise des Befehls.

b — beides: Das Flag **beinflußt** den Befehl, **und wird** durch den Befehl **gegebenenfalls geändert.**

p — passiv: Das Flag **wird gegebenenfalls** durch den Befehl **verändert.**

Als letztes haben wir die zu jedem Befehl möglichen Adressierungsarten aufgeführt, zu der auch die Symbolform, der entsprechende Opcode und die Anzahl der benötigten Bytes gehört.

10/3.1

Alphabetische Befehlsübersicht

Bezeichnungen			Funktion	Flags N	V	B	-	D	I	Z	C	Adressierung			
Befehls-kürzel	englische Bedeutung	deutsche Bedeutung										Adressierungs-art	Symbolform	Code	Bytes
ADC	Add with carry	Mit Übertrag addieren	A=A+M+C	p	p			a		p	b	unmittelbar	ADC #Op	69	2
												Zero Page	ADC Op	65	2
												Zero Page,X	ADC,X	75	2
												Absolut	ADC Op	6D	3
												Absolut,X	ADC Op,X	7D	3
												Absolut,Y	ADC Op,Y	79	3
												Vorindiziert	ADC (Op,X)	61	2
												Nachindiziert	ADC (Op),Y	71	2
AND	AND Accu with Memory	Und-Verknüpfung Speicher-Akku	A=A&M	p						p		Unmittelbar	AND #Op	29	2
												Zero-Page	AND Op	25	2
												Zero Page,X	AND Op,X	35	2
												Absolut	AND Op	2D	3
												Absolut,X	AND Op,X	3D	3
												Absolut,Y	AND Op,Y	39	3
												Vorindiziert	AND (Op,X)	21	2
												Nachindiziert	AND (Op),Y	31	2
ASL	Airthmetic Shift Left	Ein Bit nach links schieben	A=A* 2 bzw. M=M*2	p						p	p	Accumulator	ASL A	0A	1
												Zero Page	ASL Op	06	2
												Zero Page,X	ASL Op,X	16	2
												Absolut	ASL Op	0E	3
												Absolut,X	ASL Op,X	1E	3
BCC	Branch if Carry Clear	Verzweige, wenn Carry-flag=0									a	Relativ	BCC Op	90	2
BCS	Branch if Carry Set	Verzweige, wenn Carry-Flag=1									a	Relativ	BCS Op	B0	2
BEQ	Branch if Equal	Verzweige, wenn Zero-Flag=1								a		Relativ	BEQ Op	F0	2

Bezeichnungen			Funktion	Flags								Adressierung			
Befehls-kürzel	englische Bedeutung	deutsche Bedeutung		N	V	B	-	D	I	Z	C	Adressierungs-art	Symbolform	Code	Bytes
BIT	Bit Test	Bit im Speicher mit Akku prüfen		p	p					p		Zero Page	BIT Op	24	2
												Absolut	BIT Op	2C	3
BMI	Branch if Minus	Verzweige, wenn Ergebnis negativ		a								Relativ	BMI Op	30	2
BNE	Branch if Not Equal	Verzweige, wenn Zero-Flag=0								a		Relativ	BNE Op	D0	2
BPL	Branch if Plus	Verzweige, wenn Ergebnis positiv		a								Relativ	BPL Op	10	2
BRK	Break	Software-gesteuerte Unterbrechung	PC+2 auf Stapel;B=1; Status auf Stapel;I=1; PCL=($FFFE) PCH=($FFFF)			p			p			Implizit	BRK	00	1
BVC	Branch if Overflow Clear	Verzweige, wenn kein Überlauf			a							Relativ	BVC Op	50	2
BVS	Branch if Overflow Set	Verzweige, wenn Überlauf			a							Relativ	BVS Op	70	2
CLC	Clear Carry-Flag	Lösche Über-tragsbit									p	Implizit	CLC	18	1
CLD	Clear Decimal-Flag	Lösche Dezimal-arithmetik-Bit						p				Implizit	CLD	D8	1
CLI	Clear Interrupt Disable Flag	Ermögliche Unterbrechungen							p			Implizit	CLI	58	1
CLV	Clear Overflow Flag	Lösche Übertragsbit			p							Implizit	CLV	B8	1
CMP	Compare to Accumulator	Vergleiche mit Akku		p						p	p	Unmittelbar	CMP #Op	C9	2
												Zero Page	CMP Op	C5	2
												Zero Page,X	CMP Op,X	D5	2
												Absolut	CMP Op	CD	3
												Absolut,X	CMP Op,X	DD	3
												Absolut,Y	CMP Op,Y	D9	3
												Vor-indiziert	CMP (Op,X)	C1	2
												Nach-indiziert	CMP (Op),Y	D1	2

Bezeichnungen			Funktion	Flags N	V	B	-	D	I	Z	C	Adressierung			
Befehls-kürzel	englische Bedeutung	deutsche Bedeutung		N	V	B	-	D	I	Z	C	Adressierungs-art	Symbolform	Code	Bytes
CPX	Compare to X	Vergleiche mit X-Register		p						p	p	Unmittelbar	CPX #Op	E0	2
												Zero Page	CPX Op	E4	2
												Absolut	CPX Op	EC	3
CPY	Compare to Y	Vergleiche mit Y-Register		p						p	p	Unmittelbar	CPY #Op	C0	2
												Zero Page	CPY Op	C4	2
												Absolut	CPY Op	CC	3
DEC	Decrement Memory	Vermindere Wert der Speicher-zelle um 1	M=M-1	p						p		Zero Page	DEC Op	C6	2
												Zero Page,X	DEC Op,X	D6	2
												Absolut	DEC Op	CE	3
												Absolut,X	DEC Op,X	DE	3
DEX	Decrement X	Vermindere Wert des X-Registers um 1	X=X-1	p						p		Implizit	DEX	CA	1
DEY	Decrement Y	Vermindere Wert des Y-Registers um 1	Y=Y-1	p						p		Implizit	DEY	88	1
EOR	Exclusive-Or Memory with Accumulator	Akku mit Speicher Exklusiv-Odern		p						p		Unmittelbar	EOR #Op	49	2
												Zero Page	EOR Op	45	2
												Zero Page, X	EOR Op,X	55	2
												Absolut	EOR Op	4D	3
												Absolut, X	EOR Op,X	5D	3
												Absolut, Y	EOR Op,Y	59	3
												Vor-indiziert	EOR (Op,X)	41	2
												Nach-indiziert	EOR (Op),Y	51	2
INC	Memory	Erhöhe Speicherzelle um 1	M=M+1	p						p		Zero Page	INC Op	E6	2
												Zero Page, X	INC Op,X	F6	2
												Absolut	INC Op	EE	3
												Absolut, X	INC Op,X	FE	3
INX	Increment X	Erhöhe X-Register um 1	X=X+1	p						p		Implizit	INX	E8	1
INY	Increment Y	Erhöhe Y-Register um 1	Y=Y+1	p						p		Implizit	INY	C8	1

4

Bezeichnungen			Funktion	Flags								Adressierung			
Befehls-kürzel	englische Bedeutung	deutsche Bedeutung		N	V	B	-	D	I	Z	C	Adressierungs-art	Symbolform	Code	Bytes
JMP	Jump to new location	Unbedingter Sprung	PLC=(PC+1) PCH=(PC+2)									Absolute	JMP Op	4C	3
												Indirect	JMP (Op)	6C	3
JSR	Jump to new location saving return adress	Unterprogramm-aufruf	PC+2 auf Stapel PCL=(PC+1) PCH=(PC+2)									Absolute	JSR Op	20	3
LDA	Load Accu with Memory	Akku mit Speicherinhalt laden	A=M	p						p		Unmittelbar	LDA #Op	A9	2
												Zero Page	LDA Op	A5	2
												Zero Page, X	LDA Op,X	B5	2
												Absolut	LDA Op	AD	3
												Absolut, X	LDA Op,X	BD	3
												Absolut, Y	LDA Op,Y	B9	3
												Vor-indiziert	LDA (Op,X)	A1	2
												Nach-indiziert	LDA (Op),Y	B1	2
LDX	Load X with Memory	X-Register mit Speicherinhalt laden	X=M	p						p		Unmittelbar	LDX #Op	A2	2
												Zero Page	LDX Op	A6	2
												Zero page, Y	LDX Op, Y	B6	2
												Absolut	LDX Op	AE	3
												Absolut, Y	LSR Op, Y	BE	3
LDY	Load Y with Memory	Y-Register mit Speicherinhalt laden	Y=M	p						p		Unmittelbar	LDY #Op	A0	2
												Zero Page	LDY Op	A4	2
												Zero Page, X	LDY Op,X	B4	2
												Absolut	LDY Op	AC	3
												Absolut, X	LDY Op,X	BC	3
LSR	Logical Shift Right	Akkuinhalt um 1 Bit nach rechts schieben	M=M/2	p						p	p	Accumulator	LSR A	4A	1
												Zero Page	LSR Op	46	2
												Zero Page, X	LSR Op, X	56	2
												Absolut	LSR Op	4E	3
												Absolut, X	LSR Op,X	5E	3
NOP	No Operation	Leerbefehl										Implizit	NOP	EA	1
ORA	Or Accu with Memory	Akku mit Speicherinhalt „odern“	A=A v M	p						p		Unmittelbar	ORA #Op	09	2
												Zero Page	ORA Op	05	2
												Zero Page, X	ORA Op,X	15	2
												Absolut	ORA Op	0D	3
												Absolut, X	ORA Op,X	1D	3

Bezeichnungen			Funktion	Flags N	V	B	-	D	I	Z	C	Adressierung			
Befehls-kürzel	englische Bedeutung	deutsche Bedeutung										Adressierungs-art	Symbolform	Code	Bytes
												Absolut, Y	ORA Op,Y	19	3
												Vor-indiziert	ORA (Op,X)	01	2
												Nach-indiziert	ORA (Op),Y	11	2
PHA	Push Accu-mulator on Stack	Akku auf Stapel bringen										Implizit	PHA	48	1
PHP	Push Processor-status on Stack	Bringe Status auf den Stapel		a	a	a	a	a	a	a	a	Implizit	PHP	08	1
PLA	Pull Accumulator from Stack	Hole Akku vom Stapel		p						p		Implizit	PLA	68	1
PLP	Pull Processor-status from Stapel	Hole Status vom Stapel		p	p	p	p	p	p	p	p	Implizit	PLP	28	1
ROL	Rotate Left One Bit	Akku bzw. eine Speicherzelle ein Bit nach links rotieren		p						p	b	Accumulator	ROL A	2A	2
												Zero Page	ROL Op	26	2
												Zero Page, X	ROL OP,X	36	2
												Absolut	ROL Op	2E	3
												Absolut, X	ROL Op,X	3E	3
ROR	Rotate Right One Bit	Akku 1 bzw. eine Speicher-zelle ein Bit nach rechts rotieren		p						p	b	Accumulator	ROR A	6A	2
												Zero Page	ROR Op	66	2
												Zero Page, X	ROR Op,X	76	2
												Absolut	ROR Op	6E	3
												Absolut, X	ROR Op,X	7E	3
RTI	Return from Interrupt	Rückkehr von Unterbrechung		p	p	p	p	p	p	p	p	Implizit	RTI	40	1
RTS	Return from Subroutine	Rückkehr von Unterprogramm	PC vom Stapel holen; PC=PC+1									Implizit	RTS	60	1
SBC	Subtract Memory from Accu	Speicher vom Akku abziehen		p	p			a		p	b	Unmittelbar	SBC #Op	E9	2
												Zero Page	SBC Op	E5	2
												Zero Page, X	SBC Op,X	F5	2
												Absolut	SBC Op	ED	3
												Absolut, X	SBC Op,X	FD	3
												Absolut, Y	SBC Op,Y	F9	3

Bezeichnungen			Funktion	Flags N	V	B	-	D	I	Z	C	Adressierung			
Befehls-kürzel	englische Bedeutung	deutsche Bedeutung		N	V	B	-	D	I	Z	C	Adressierungs-art	Symbolform	Code	Bytes
												Vorindiziert	SBC (Op,X)	E1	2
												Nachindiziert	SBC (OP),Y	F1	2
SEC	Set Carry Flag	Übertrags-Bit setzen									p	Implizit	SEC	38	1
SED	Set Decimal Mode	Dezimal-arithmetik einschalten						p				Implizit	SED	F8	1
SEI	Set Interrupt Disable Bit	Verhindere Unterbrechungen							p			Implizit	SEI	78	1
STA	Store Accumula-tor in Memory	Akku speichern										Zero Page	STA Op	85	2
												Zero Page, X	STA Op,X	95	2
												Absolut	STA Op,X	8D	3
												Absolut, X	STA Op,X	9D	3
												Absolut, Y	STA Op,Y	99	3
												Vor-indiziert	STA (OP,X)	81	2
												Nach-indiziert	STA (OP),Y	91	2
STX	Store X in Memory	X-Register speichern										Zero Page	STX Op	86	2
												Zero Page, Y	STX,Y	96	2
												Absolut	STX Op	8E	3
STY	Store Y in Memory	Y-Register speichern										Zero Page	STY Op	84	2
												Zero Page, X	STY,X	94	2
												Absolut	STY Op	8C	3
TAX	Transfer Accu to X	Übertrage Akku ins X-Register		p						p		Implizit	TAX	AA	1
TAY	Transfer Accu to Y	Übertrage Akku ins Y-Register		p						p		Implizit	TAY	A8	1
TSX	Transfer Stack-pointer to X	Übertrage Stapelzeiger ins X-Register		p						p		Implizit	TSX	BA	1
TXA	Transfer X to Accu	Übertrage X-Register in den Akku		p						p		Implizit	TXA	8A	1
TXS	Transfer X to Stackpointer	Übertrage X-Register in den Stapelzeiger										Implizit	TXS	9A	1
TYA	Transfer Y to Accu	Übertrage Y-Register in den Akku		p						p		Implizit	TYA	98	1